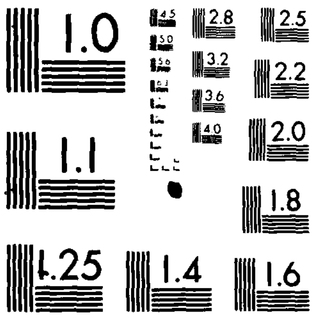
AD-A116 237 HUGHES RESEARCH LABS MALIBU CA F/G 9/2
ADVANCED IMAGE UNDERSTANDING.(U)
DEC 81 G R NUDD, S D FOUSE F33615-80-C-1080
UNCLASSIFIED AFWAL-TR-82-1095 NL

1 OF 2
AD
A116237


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

AFWAL-TR-82-1095

A[illegible] IMAGE UNDERST[illegible]

G. R. Nudd and S. D. Fouse

Hughes Research Laboratories
3011 Malibu Canyon Road
Malibu, CA 90265

Interim Report for Period
1 April 1980 through 30 September 1981

December 1981

Approved for Public Release; Distribution Unlimited

Avionics Laboratory
Air Force Wright Aeronautical Laboratories
Air Force Systems Command
Wright-Patterson Air Force Base, Ohio 45433

Louis A. Tamburino

LOUIS A. TAMBURINO
Project Engineer

Donald L. Moon

DONALD L. MOON, Chief
Information Processing Technology Branch
Avionics Laboratory

FOR THE COMMANDER

Richard H. Boivin

RICHARD H. BOIVIN, Colonel, USAF
Chief, System Avionics Division
Avionics Laboratory

Accession For	
NTIS GRA&I	✓
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A	

DTIC COPY INSPECTED 2

[illegible] 82 1300

4

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFWAL-TR-82-1095	2. GOVT ACCESSION NO. AD-A116 237	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) ADVANCED IMAGE UNDERSTANDING		5. TYPE OF REPORT & PERIOD COVERED Interim Technical Report 1 April 1980 - 30 Sept 1981
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) G. R. Nudd, S. D. Fouse		8. CONTRACT OR GRANT NUMBER(s) F33615-80-C-1080
9. PERFORMING ORGANIZATION NAME AND ADDRESS Hughes Research Laboratories 3011 Malibu Canyon Road Malibu, CA 90265		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 3119 00 02
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, Virginia 22209		12. REPORT DATE December 1981
		13. NUMBER OF PAGES 107
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Air Force Wright Aeronautical Laboratories Wright-Patterson Air Force Base (AFWAL/AART) Dayton, Ohio 45433		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Convolution, residue-based arithmetic, residue-based processor, image understanding, texture, line finder.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report presents the work that has been performed by Hughes Research Laboratories on the [illegible] Image Understanding contract from April 1, 1980 to September 30, 1981. [illegible] work is primarily concerned with the development of hardware to allow real-time operation of the image understanding algorithms which are also being developed under the same contract.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

ASD 82 1300

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

FIGURE PAGE

SUMMARY

The following represents the work undertaken by Hughes Research Laboratories, Malibu, California in support of the Image Processing Institute of the University of Southern California. The work was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by the Air Force Avionics Laboratory at Wright Patterson Air Force Base, Ohio, under contract F33615-80-C-1080, ARPA Order No. 3119. The work period represented by this report (in three separate sections) is 1 April 1980 through 30 September 1981.

PROGRESS REPORT

April 1980 to October 1980

SECTION 1

INTRODUCTION

The main emphasis of this program was the investigation of the impact and potential benefit of very large scale integration (VLSI) and high-density IC technologies for image understanding. The task was somewhat wider than our previous work for the Image Understanding Program, where we successfully developed special-purpose high-speed primitives for the low-level processing operations.

The progress that is currently being made in silicon technology and the development of more sophisticated algorithms for image understanding has provided the basis for a major advance in processing capability. The work described here was undertaken at Hughes Research Laboratories (HRL), where we are actively involved in the development of image analysis and understanding software for applications such as terminal homing, image bandwidth compression, and scene matching. We also have numerous active research programs in micro-electronic technology. The people involved in this program were also involved in the VHSIC-0 program. In addition, at HRL we have eleven VHSIC-3 technology development programs. We are, therefore, aware of the major developments in these and other important programs, and where appropriate, we have coordinated our efforts.

Our major emphasis has been to review numerous complex image-understanding algorithms and devise VLSI concepts for them. This includes understanding the processing, data flow, timing, and storage requirements. We have also performed a detailed partitioning of potential VLSI chips based on total gate count, silicon area, communication requirements, and power considerations. From this work, three potential systems have emerged: a line finder, texture analyzer, and segmenter. The details of these systems are included in this report, together with the necessary gate estimate and parts count for VLSI chips (Tables 1 and 2).

In addition to this, we have, through our detailed analysis of the VLSI processor requirements and configuration, identified a highly modular programmable digital processing element RADIUS (residue-based arithmetic image-understanding system). These concepts have been developed in state-of-the-art

Table 1. Gate Count for Texture Classification System

5 line kernel generation	15K gates
5 x 5 convolution	27K gates
5 x 5 variance	10K gates
Normalization	1K/channel
Large window statistical calculation	8.25K/channel
Transform (M input channels)	2.1K·M/output channel

Table 2. Gate Count for Line Finder System

Edge detection	178K
Thinning	190 gates
Edge linking	500
Edge tracing	12 Mbit memory + 5K logic gates

n-MOS technology, and we anticipate that the first demonstration will be available in the early part of 1982. Since RADIUS is based on residue arithmetic, it can perform a wide variety of processing operations (including variance and moment calculation) and all convolutions and local area operations with very high circuit function density. The modularity of our approach allows rapid and easy design in VLSI and provides programmability and portability.

In addition, a major emphasis of all our work has been to ensure that the hardware concepts and changes will be fully integratable with existing commercial hardware, such as the DEC mainframes. To this end, we are developing the necessary interfaces to the RADIUS processor so that it can be used as an attached processor to the DEC series, communicating through the DEC-UNIBUS.

SECTION 2

METHOD

Our objective for this project has been to provide some general results that will allow the image understanding community to take full advantage of VLSI technology as it becomes available. We expect our results to include definitions for several special-purpose chips that will have wide applicability to IU systems. The question to be answered then is what types of functions should be cast into a VLSI chip. The goals of this program are analogous to the on-going VHSI efforts where a search continues for commonality across a broad range of DOD systems. Here we have taken a longer-range view specifically for image understanding. The project has been coordinated with the Hughes VHSIC program and shares many goals with it. The major difference is that we would like to see commonality across IU systems. We are tracking the VHSIC program, and, where appropriate, we will be able to take advantage of the work being done there that complements our own effort.

The area of this study has necessarily been somewhat broad. It has differed considerably from our previous work where we developed special-purpose high-speed primitives for IU. Our aim this period has been to stand back and take a broad view of IU requirements and to attack the question of what special-purpose VLSI functions would be appropriate for IU systems that will not be developed under present or foreseeable VHSIC programs. Our approach has been to

- Select three representative systems to study and characterize;
- Perform a commonality study across the three systems;
- Do logic designs for each system;
- Partition the designs onto VLSI chips;
- Test and refine designs using simulation techniques;
- Identify relevant system parameters;
- Generalize to additional systems and refine function partitioning.

So far, we have selected three systems and performed a preliminary commonality study, identifying one subsystem as a likely candidate for a VLSI chip. This is a local area processor which will perform sliding window arithmetic operations including convolution, variance, and moment calculations. We have designed a processor based on the residue arithmetic technique that should provide significantly better performance for this type of operation than a binary processor. In addition, we have performed logic designs on two of the three systems selected: the line finder and the texture classification systems. What remains to be done is to design the third system, the segmenter, refine our designs using results from simulations, and then extend the results to additional IU systems.

SECTION 3

RESIDUE-BASED ARITHMETIC IMAGE UNDERSTANDING SYSTEM (RADIUS)

Almost all of the systems we have looked at require some sort of local-area processing where the output pixel is a function of the input pixel and its M nearest neighbors, where M is usually 8, 24, 48, etc. The function takes numerous forms, but typically is either arithmetic or logical, or a combination of both. If the function is arithmetic and does not require division or absolute value, then residue arithmetic techniques can be used. Examples where these conditions are met can be found in two of the systems we are studying: the line finder and the texture classification systems. The line finder detects edges by convolving the image with six 5 x 5 masks, each mask responding to a different direction. Since the convolution operation requires multiplications and additions, this is easily done using residue techniques. Similarly, the texture classification system convolves the input image with several 5 x 5 masks.

An arithmetic local area processor seems to be a natural choice for a subsystem that can take advantage of VLSI technology. This is because of the complexity of the logic for doing multiples. Using current technology, a high-speed multiplier requires an entire chip (such as the TRW 10-MHz multiplier). One approach being considered for real-time hardware is to compromise on the coefficients and thereby reduce the complexity of the hardware. The residue techniques can offer significant advantages without compromise, since multiplication can be performed using look-up tables; these tables will be small because the bases used will be small. As an illustration of this, Figure 1 shows the components involved in RADIUS. The input data are initially encoded into their equivalent representation in each base. This is most easily done using a ROM look-up table with an address space equal to the input dynamic range, and a bit depth equal to the number of bits required to represent the Base-1. The actual computation is then performed by the local area processors using one processor per base. The processor output word size is identical to

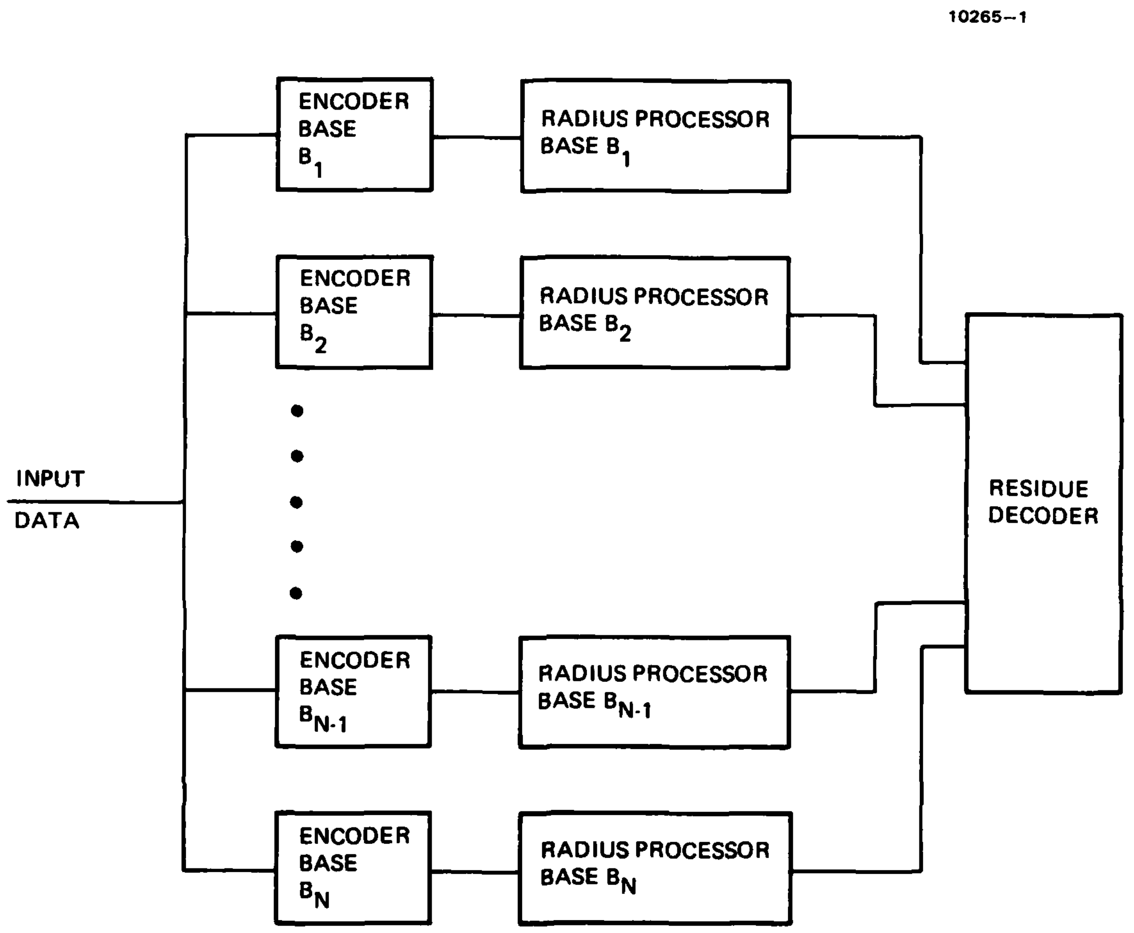

10265–1
INPUT
DATA
ENCODER BASE B_1
ENCODER BASE B_2
ENCODER BASE B_{N-1}
ENCODER BASE B_N
RADIUS PROCESSOR BASE B_1
RADIUS PROCESSOR BASE B_2
RADIUS PROCESSOR BASE B_{N-1}
RADIUS PROCESSOR BASE B_N
RESIDUE DECODER

Figure 1. A residue-based processor.

the input word size with no loss in accuracy. This is due to the modular nature of residue arithmetic. The data out of the convolvers then goes into a residue-to-binary decoder. This block can be composed of a very large look-up table or a system composed of ROMs and adders. The exclusive use of look-up tables is well suited to VLSI since these are highly regular structures and hence can be made to be very dense and inexpensive.

Many arithmetic operations can be performed with table look-ups; but without the residue concept, this approach is typically not feasible, as the tables required become overwhelmingly large. Since both the line-finder system and the texture system utilize 5 x 5 convolutions, the processor system should be capable of being used to perform 5 x 5 convolutions with programmable weights. The dynamic range capabilities of the system should allow for 6-bit input and 6-bit kernel weights for a total output dynamic range of 17 bits. In addition, the system should be able to operate at a 10-MHz data rate. The dynamic range of a residue computation system is determined by the product of all the bases. Using 31, 29, 23, and 19 as bases will give us a dynamic range of 392,863, which exceeds 2^{17}. The bases will have other benefits since they are all prime numbers.

Figure 2 shows a block diagram for a 5 x 5 local area processor for a single base. There are five data inputs, one for each line of the 5 x 5 area. Input data is put into a register which is the first element of a five-element shift register. When new data is transferred in, the previous data is transferred to the next register, and so on. There are five inputs, and thus, five shift registers with five elements each, for a total of 25 registers. The contents of each of the 25 registers are used to address 32-element by 5-bit RAMs. These RAMs are look-up tables for the multiplication of the input data and the kernel weight. The outputs of the 25 RAMs are then added by a tree of 24 residue (or modular) adders. A residue adder calculates the (A + B) mod base.

There are several noteworthy points about this design. First, only 5 bits come out of the multiply or add. This is because these operations are cyclical, and the output values are in the same range as each of the input values. Second, the only part of the processor that is dependent on the base is the residue adder. Since the multiplier must be programmed for the weights anyway,

10265-2

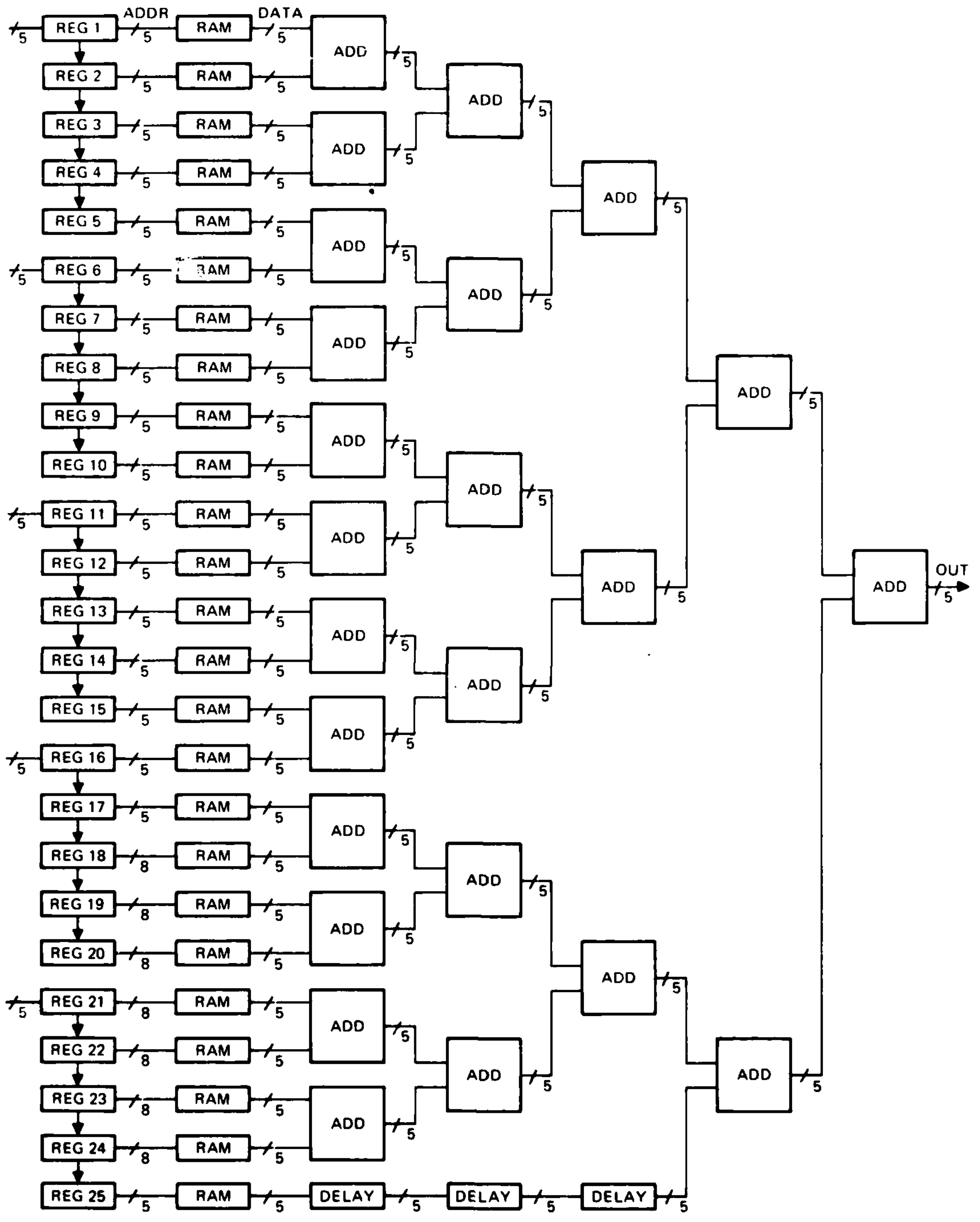


Figure 2. 5 x 5 local area processor.

the choice of base is also programmed at the same time. And finally, concerning the programming of the RAMs, a bidirectional data bus will be provided on the data lines of the RAMs to allow programming as well as to serve as a test point in the processor. Additional control lines into a binary decoder will be required for programming and controlling the bus for testing. A possible design for the residue adder is shown in Figure 3. Essentially, the device performs a binary addition of the two operands, compares the result to the base, and subtracts the base if the result is greater than or equal to the base. The adder is programmed by providing the base value to the comparator and the two's complement value of the base to the second adder. The programming can be done in one of two ways. First, and preferable, is to provide a register on the chip for the base value and another register for the two's complement of the bases. The value in the register must then be made available to each of the residue adders that will be on the chip. The second alternative is to make the base programmable by using a ROM. This has the advantage that base values can be stored directly in the adder, which will prevent possible routing problems.

We are developing a chip based on RADIUS. However, to reduce the cost of development and the associated risk, we are restricting the chip to a 5 x 1 area processor. If this demonstration is successful, we will be able to use the designs and digitized data base as a common module to be prepared across an entire VLSI chip, resulting in a very high density, high throughput processor. Figure 4 shows the design for a 5 x 1 processor, and Figure 5 shows the configuration of a 5 x 5 convolution system which utilizes the 5 x 1 processor chip. The data are encoded using a 64-element by 20-bit ROM. The kernel is generated using four 20-bit wide line delays, 5 bits for each base. The output of the line delays is input directly to an array of 20 5 x 1 processors, five for each base. For each base, four 1,024 x 5-bit ROMs are used to sum the results of the five rows. Note that conventional adders cannot be used since these additions must be done modularly, just as on the chip. Finally, 5 bits from each base convolver is input to a decoder network, which will be an array of ROMs and possibly some logic. The output of the decoder is then a 17-bit binary word.

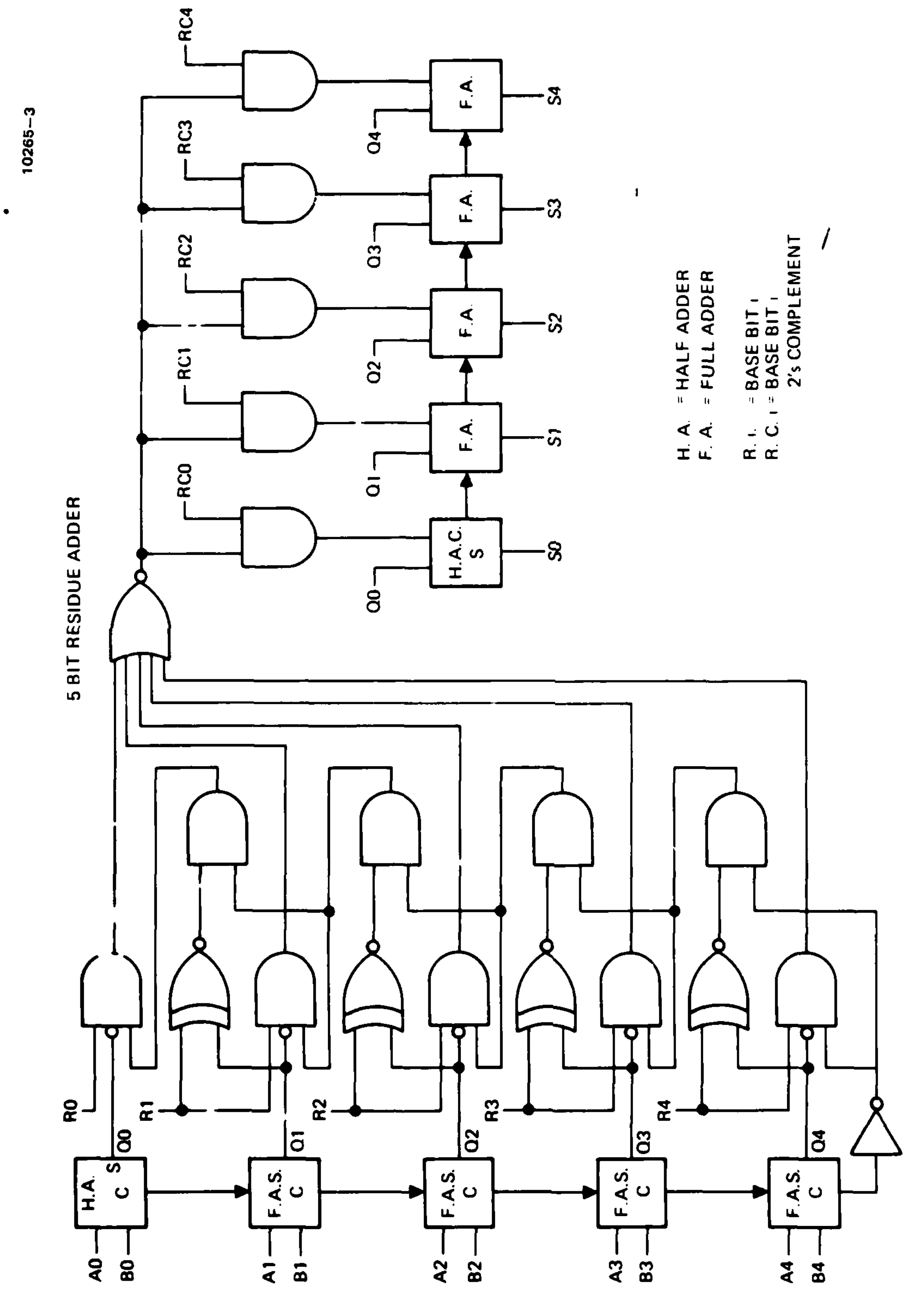


Figure 3. Residue adder.

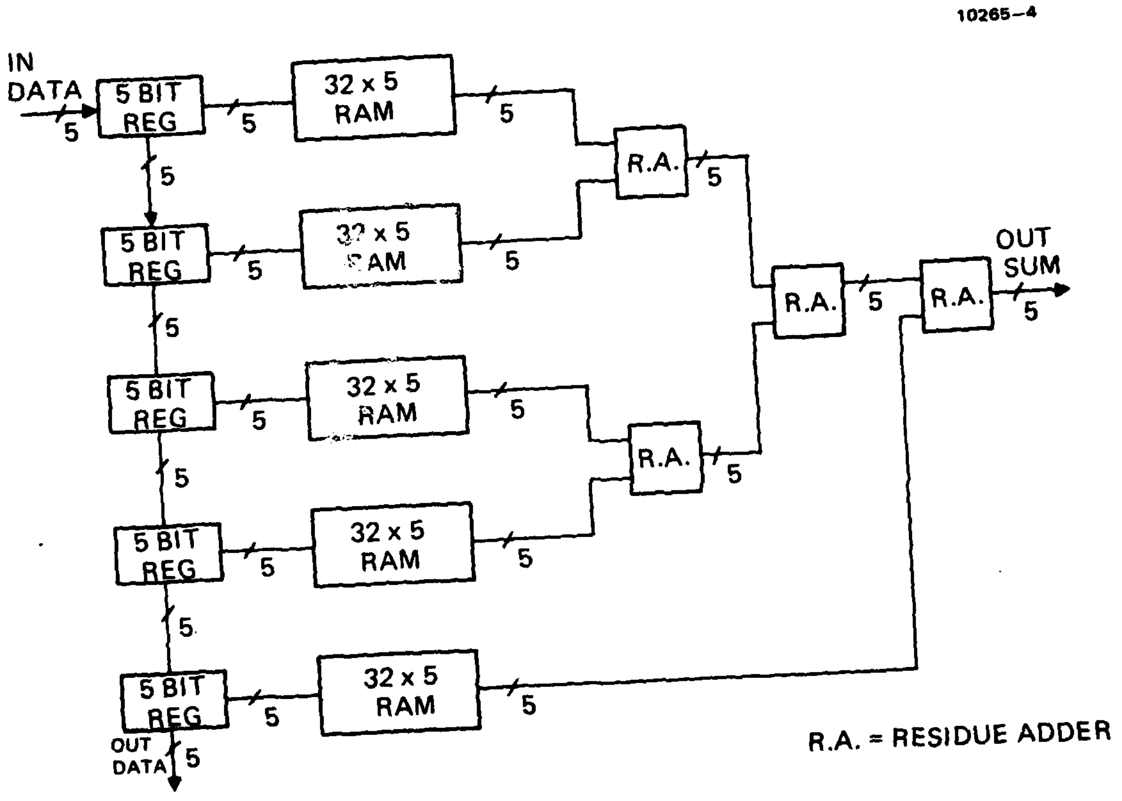


Figure 4. 5 x 1 RADIUS processor chip.

10265–5

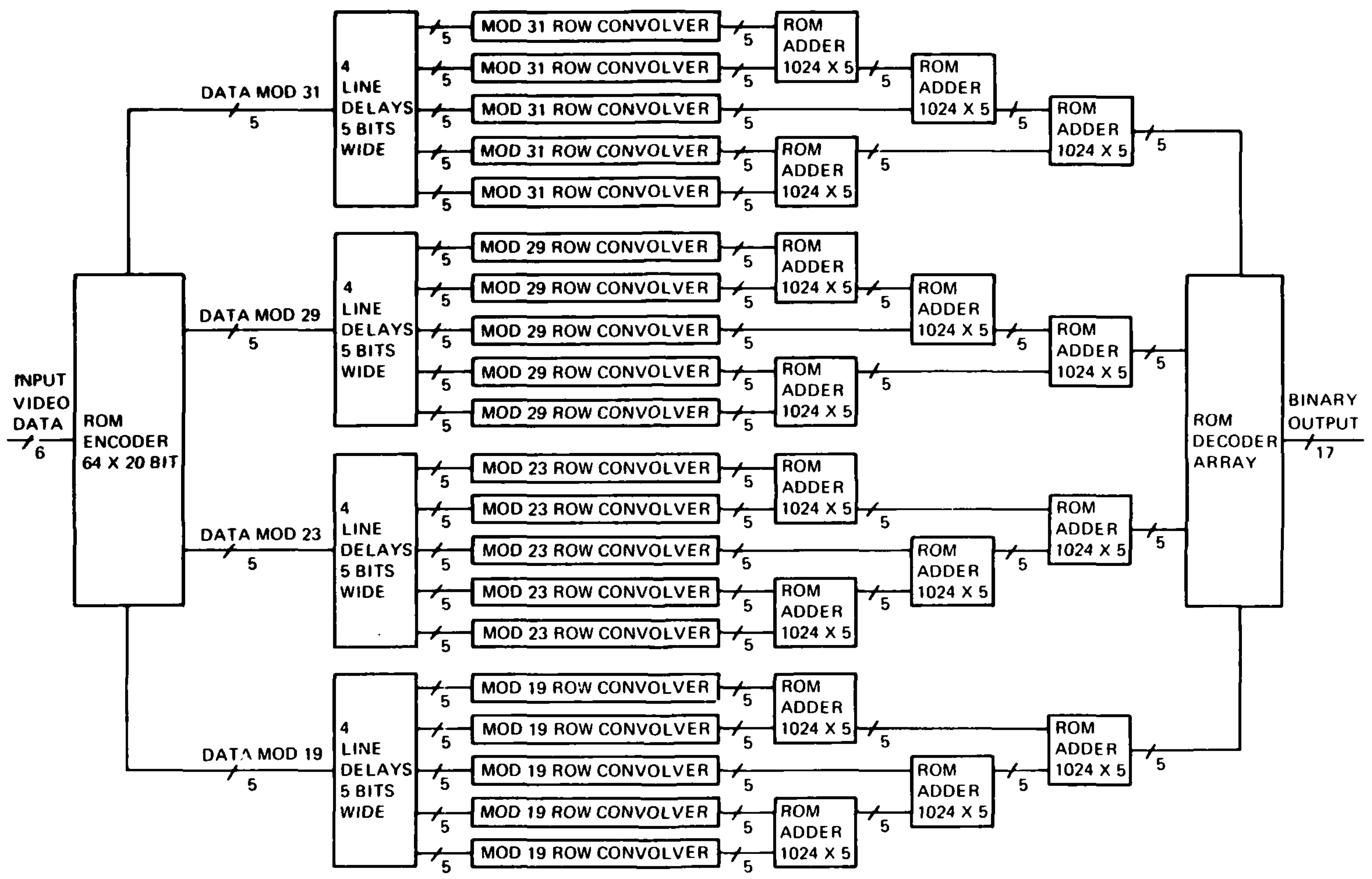


Figure 5. A 5 x 5 RADIUS processor using 5 x 1 chips.

SECTION 4

TEXTURE CLASSIFICATION SYSTEM

As mentioned earlier, logic design was performed for both the texture classification system and the line finder system. This section describes the design generated for the texture system. Figure 6 illustrates the data flow of the texture system. The input video is processed in N independent channels, where each channel generates an element of a feature vector. This means that for each scalar pixel input there is a vector value output, the length of the vector being equal to N, the number of channels in the system. The processing done in the channels involves a small window convolution, a normalization step, and a large window energy measure. The outputs are combined to form a vector, which is then transformed using a linear transformation. The elements of the transformed vector are used to evaluate a discriminant function, the value of which is used to perform the classification. Alternatively, the transformed feature vector can be input to a segmentation routine.

Six major functions must be performed:

- Small window convolution
- Small window statistical calculation
- Scaling
- Large window statistical calculation
- Linear transformation
- Discriminant function evaluation.

The operations involved in each of the functions are discussed below, along with the hardware required to perform the functions.

A. SMALL WINDOW CONVOLUTION

The first step in the processing involves computing a 5 x 5 convolution. The design of a system (RADIUS) to perform this computation using the technique of residue arithmetic is presented in the previous section. The basic block that performs the 5 x 1 convolution is now being designed as an NMOS chip (CRC 181) at the Hughes Carlsbad Research Center.

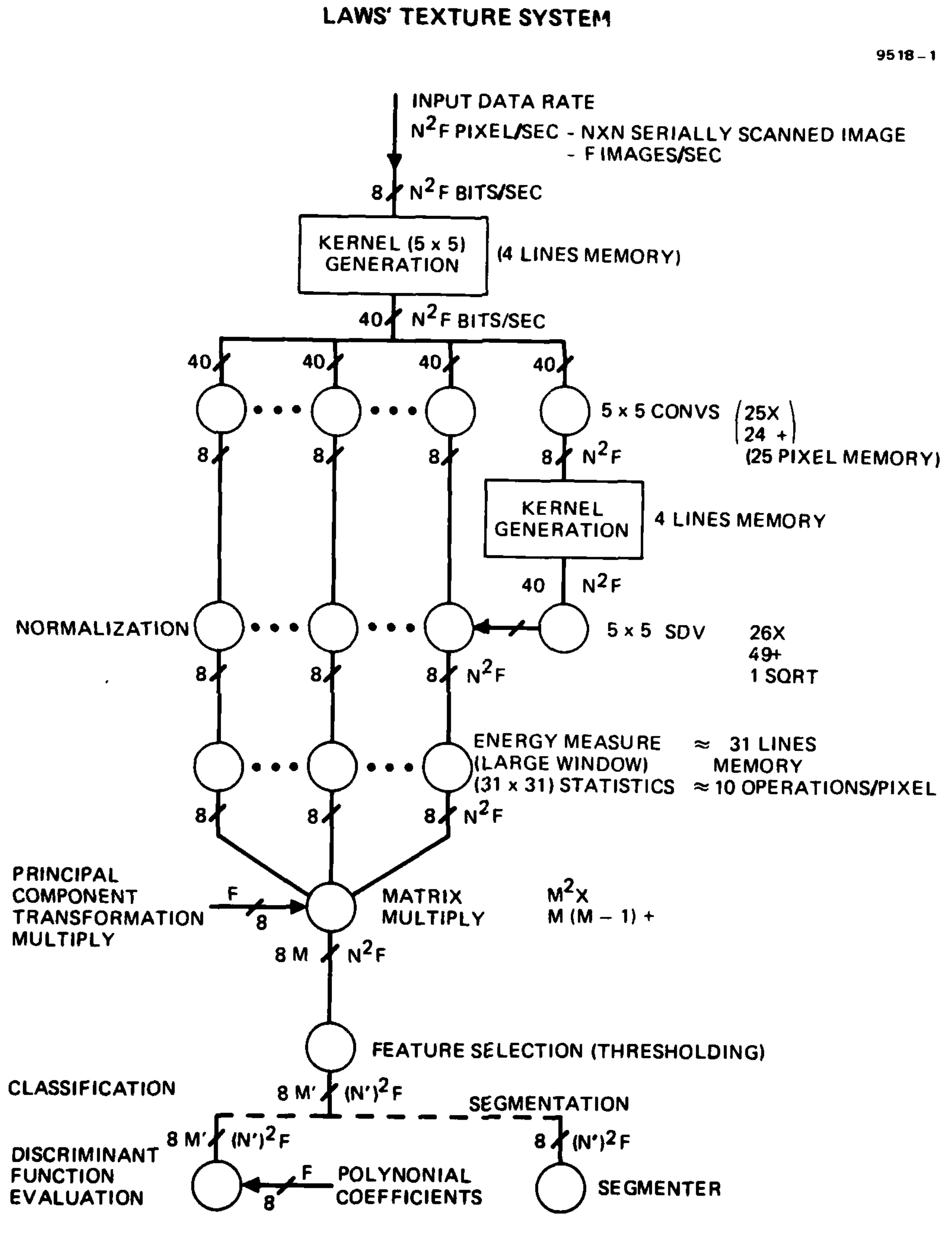


Figure 6. Laws texture classifier.

B. SMALL WINDOW STATISTICAL MEASURE

A general structure for calculating a statistical moment over a two-dimensional window was described in Hughes invention disclosure PD80078; Figure 7 shows the specific structure for calculating the variance over a 5 x 5 window. This architecture assumes that the image data are being input in raster scan format. As each pixel is introduced into the system, the window that is being processed is shifted one column to the right, dropping the left-most column and adding the column on the right. This means that the function for the new window position can be calculated by subtracting the contribution of the lost column and adding the contribution from the new column. It should be noted, as illustrated in Figure 8, that as the center of the 5 x 5 window moves across the image each subsequent kernel can be formed by the removal of a single pixel at the top right, for instance, and the addition of one new pixel at the bottom left. With this proviso, we can scan the image directly by successively eliminating the 5 pixel column at the trailing edge and adding the new column on the right as shown. Figure 8 illustrates this method for updating the window function.

This technique greatly reduces the necessary data bandwidth for calculating the mean and the sum of squares for the processing window; these can then be combined to form the variance. The structure shown includes shift registers for pixel storage, column storage for the mean calculation, and the column storage for the sum of squares calculation. The only arithmetic logic required for this structure is four 3-input adders and three multipliers.

This is a function that could be implemented using a residue technique, since only arithmetic functions are being performed. Since the preceding function is already being performed using residue arithmetic, the conversion back to binary could be done after the moment calculation. The structure would look identical, but the blocks themselves would each be smaller. For example, with a smaller wordsize the memory would be reduced. Also, the multiplier box could be replaced with look-up tables, thus providing a savings in hardware. To determine the feasibility, a statistical dynamic range analysis will be performed to see how many bases would be required and thus if the residue technique could provide normalization.

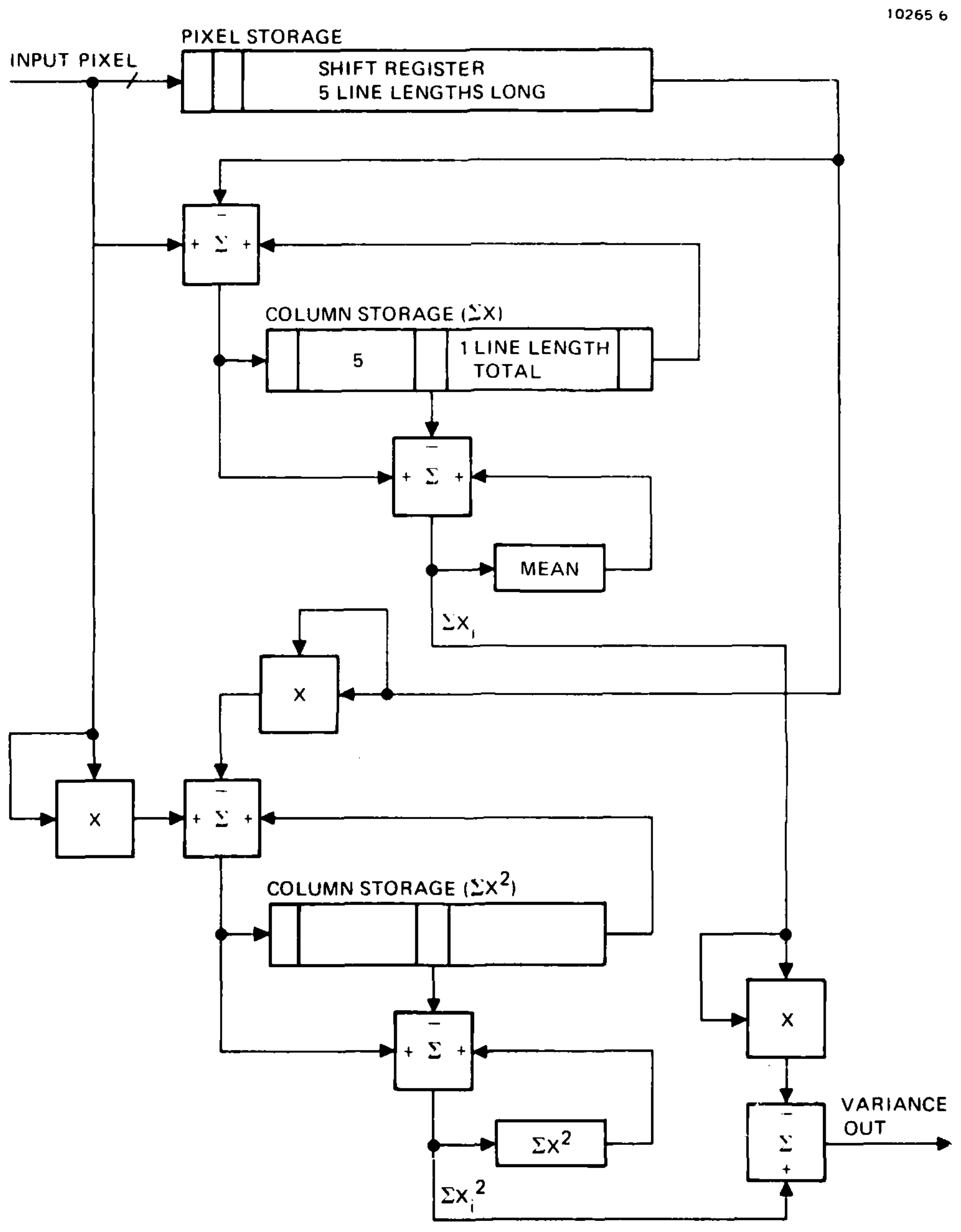


Figure 7. 5x5 variance calculation.

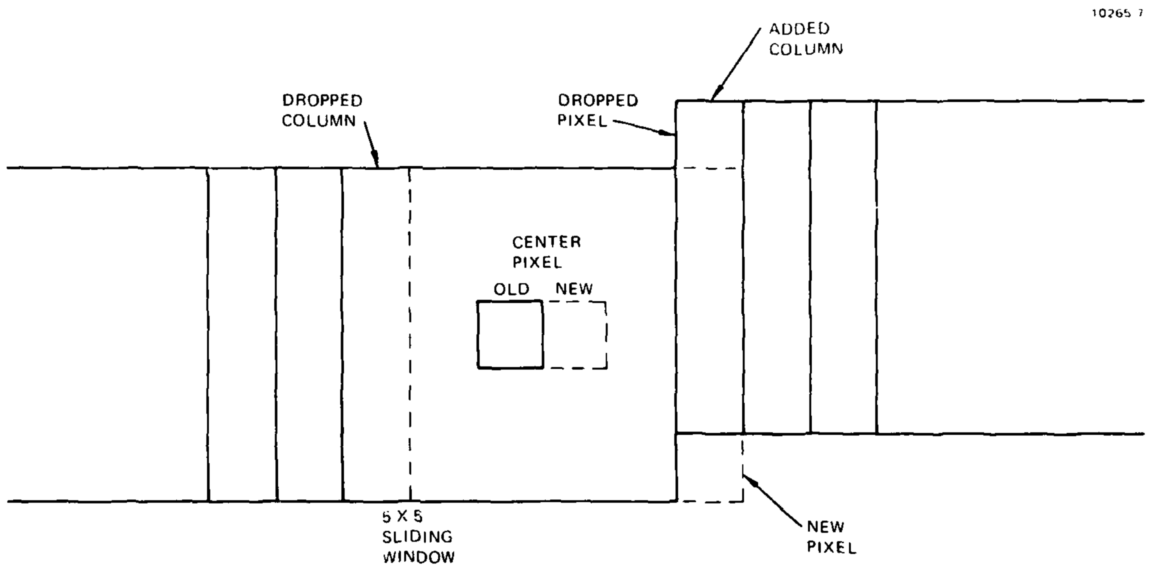


Figure 8. Sliding window updating.

C. NORMALIZATION

The next major function to be performed is a division or normalization function. The output of the convolutions are divided by the output of the small window moment calculation, on a pixel by pixel basis. This requires that there be some memory to delay the output of the convolutions while the moment is being calculated. The memory required would be approximately 5 line lengths x 8 bits x N channels.

A pipelined system for performing an integer divide is shown in Figure 9. This performs a division between two binary numbers using the direct method: an 8 bit divisor would require 8 stages. If more precision were required, more stages could be used, and these would calculate the fractional part of the answer. The direct method was chosen over the iterative method because of the synchronous nature of this system.

D. LARGE WINDOW STATISTICAL CALCULATION

The structure for performing the large window statistical calculation is the same as for the small window calculation. The function suggested by Laws is the sum of the absolute values of the pixels.

Figure 10 shows a structure fc, performing this function. As suggested in the discussion of the small window moment calculation, this structure could be implemented in residue. In fact, if the normalization step could be skipped, then the whole system could be accomplished in residue. There are two problems, however:

- An absolute value cannot be accomplished in residue, and thus the next best thing would be to use the sum of squares.

- A preliminary dynamic range analysis using the sum of squares over a 15 x 15 window indicates that the system would need to support a dynamic range in excess of 36 bits. This would definitely be prohibitive in that a large number of bases would be required and the bases themselves would require over 5 bits to encode.

It may be that a statistical dynamic range analysis will show that it is possible to achieve a low probability of overflow with a more reasonable dynamic range (e.g., <30 bits).

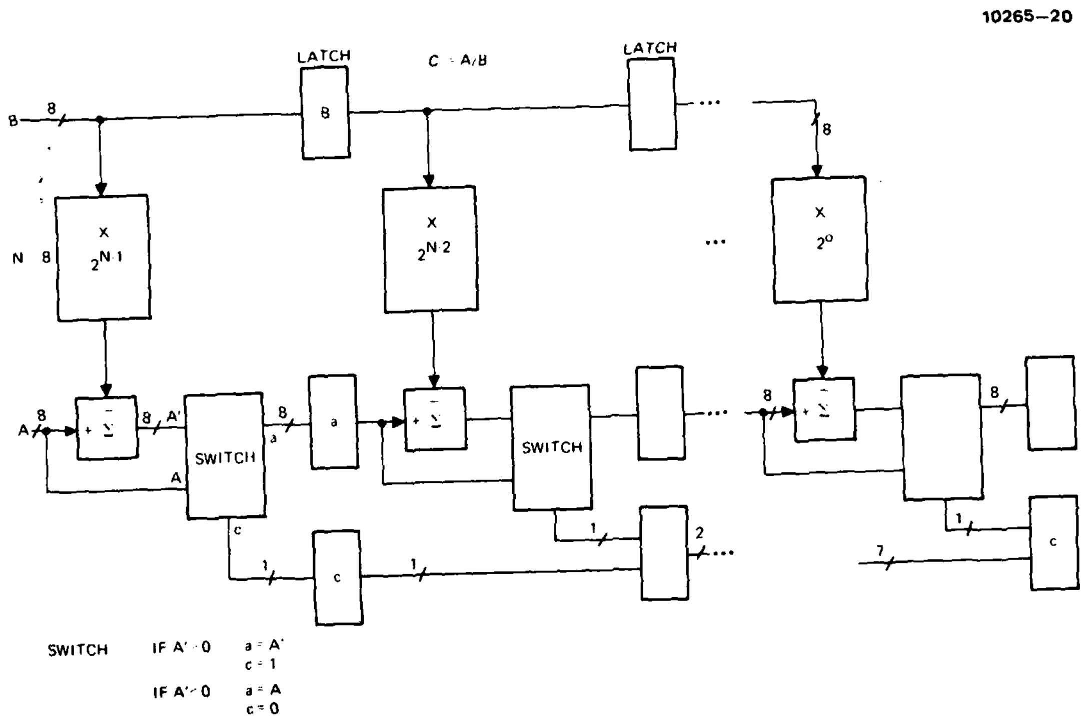


Figure 9. Structure for integer divide.

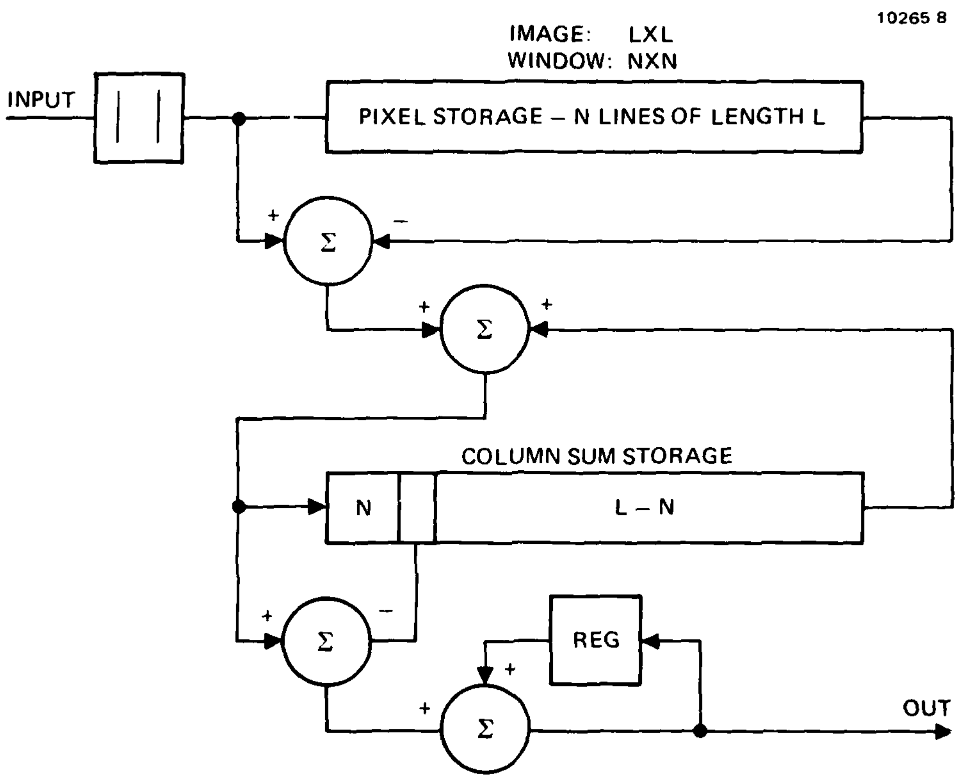


Figure 10. Large window energy measure ($\Sigma|x|$).

E. LINEAR TRANSFORMATION

This function will generate a vector with M components by forming linear combinations of the N components of the input vector. Since the structure shown in Figure 11 can be used to generate a single component of the output vector, the entire output vector can be generated by replicating this structure M times. This structure is basically the same as that used for the convolution. It is composed of N registers, N multipliers, and N-1 adders. As with the convolution, the value of the linear weights will be programmable, with the actual mechanism for programming depending on the structure of the multiplier (either memory look-up or logic). The only difference would be in the format of the data input. For the convolution, the data are shifted through the registers; but for the linear transformer, the registers are all loaded in parallel.

F. DISCRIMINANT FUNCTION EVALUATION

No hardware was designed for this processing step since the form of the function is unknown. If the function is linear, then the structure used in the previous step could also be used for this one. Any other form, such as a higher-order polynomial, would require additional multipliers and adders.

G. GATE COUNT TABULATION

Table 1 summarizes the results of the texture system design, presenting the number of gates required for each function. These data will be used later in the study for the VLSI partitioning.

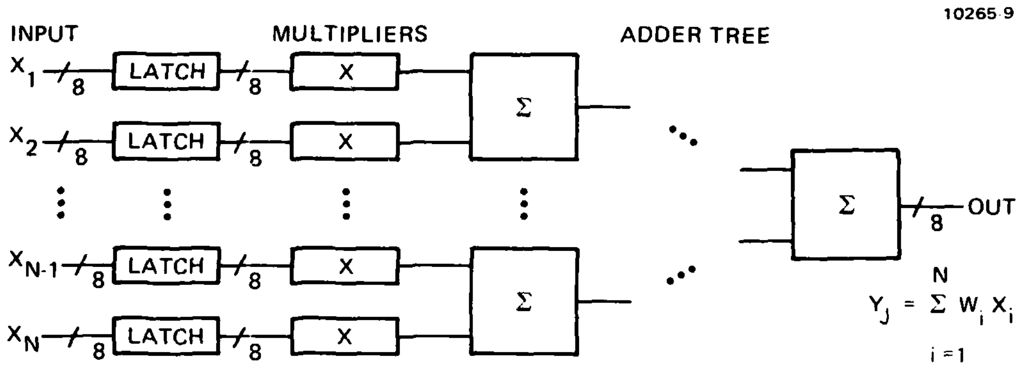


Figure 11. Linear transformation structure.

SECTION 5

LINE FINDER SYSTEM

This section describes the logic design of the line finder system. Figure 12 shows the data flow graph for the system and identifies four major functions:

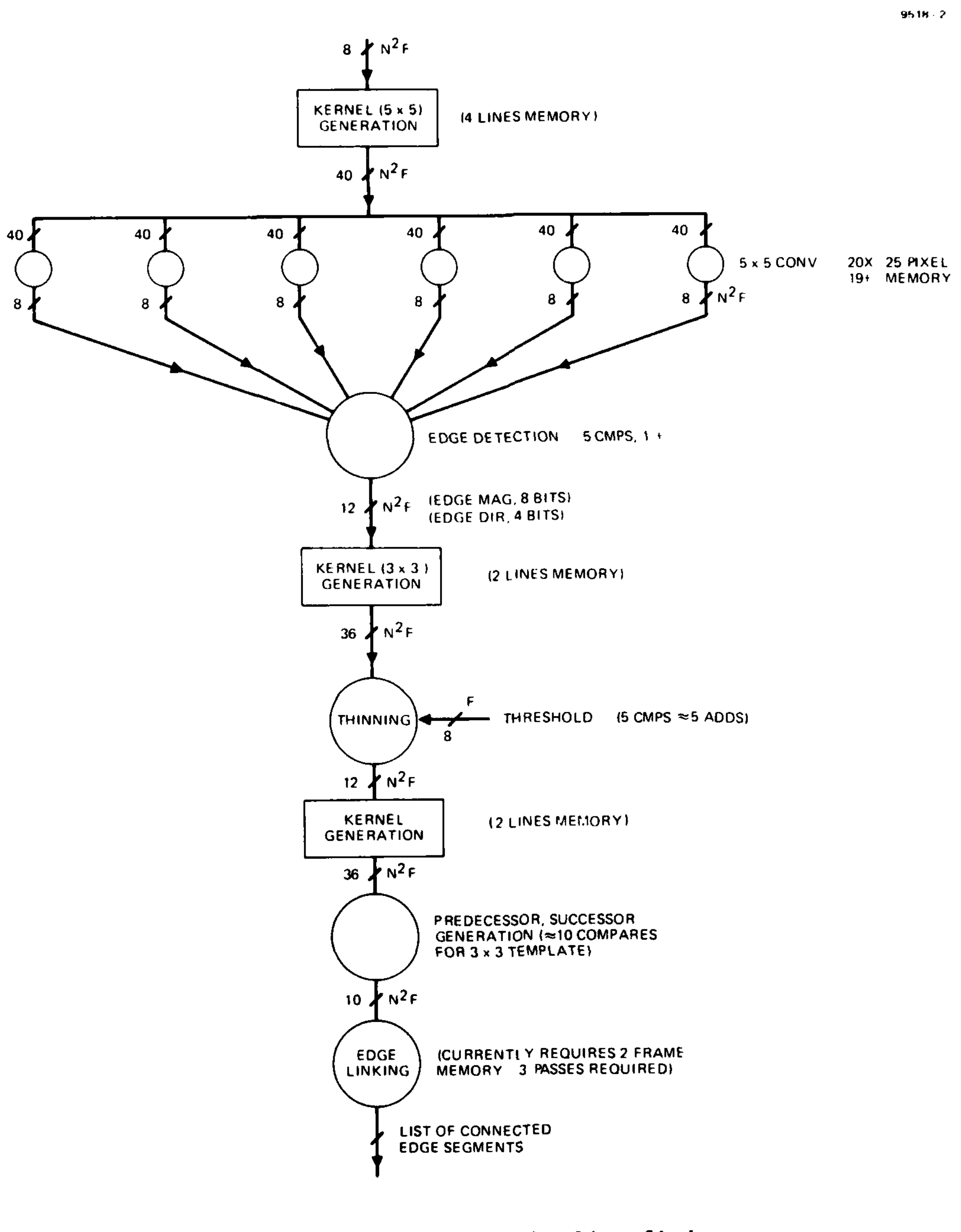
- Edge detection
- Edge thinning
- Edge linking
- Edge tracing.

A design is presented for each of the four functions, and an estimated gate count is given.

A. EDGE STRENGTH COMPUTATION AND DETECTION

Edge strength computation is performed by convolving 5 x 5 masks in six directions with the image of interest (Figure 13). The mask directions are set at 30° intervals, and the weights are shown in Figure 14. Following edge strength computation, the magnitudes from the six directions are compared, and the direction with greatest magnitude is selected as the edge vector for the pixel location.

There is some question as to what the optimum mask size, weights, and number of mask directions should be. For example, a larger mask size (say 5 x 5) is more immune to noise, but takes considerably more hardware to implement (than, say, a 3 x 3 mask). The same can be said for implementing the masks in six directions as opposed to four. It would be desirable from a hardware point of view to use a small mask size and as few mask directions as possible without sacrificing too much on performance. RADIUS can be used to perform the six convolutions. Following these six convolutions, direction tags will be added to the edge magnitudes (see Figure 13). Each data word consists of 12 bits at this stage, 8 bits for magnitude, and 4 bits for direction. The magnitude part of each direction is then compared with that of the other directions, until the direction with greatest magnitude is found. Five comparators are needed to perform the magnitude comparisons.



9518-2
8 N²F
KERNEL (5 x 5) GENERATION
(4 LINES MEMORY)
40 N²F
40
8
5 x 5 CONV
20X 25 PIXEL
19+ MEMORY
8 N²F
EDGE DETECTION 5 CMPS, 1 +
12 N²F
(EDGE MAG, 8 BITS)
(EDGE DIR, 4 BITS)
KERNEL (3 x 3) GENERATION
(2 LINES MEMORY)
36 N²F
THINNING
F
8
THRESHOLD (5 CMPS ≈5 ADDS)
12 N²F
KERNEL GENERATION
(2 LINES MEMORY)
36 N²F
PREDECESSOR, SUCCESSOR GENERATION (≈10 COMPARES FOR 3 x 3 TEMPLATE)
10 N²F
EDGE LINKING
(CURRENTLY REQUIRES 2 FRAME MEMORY 3 PASSES REQUIRED)
LIST OF CONNECTED EDGE SEGMENTS

Figure 12. Nevatia-Babu line finder.

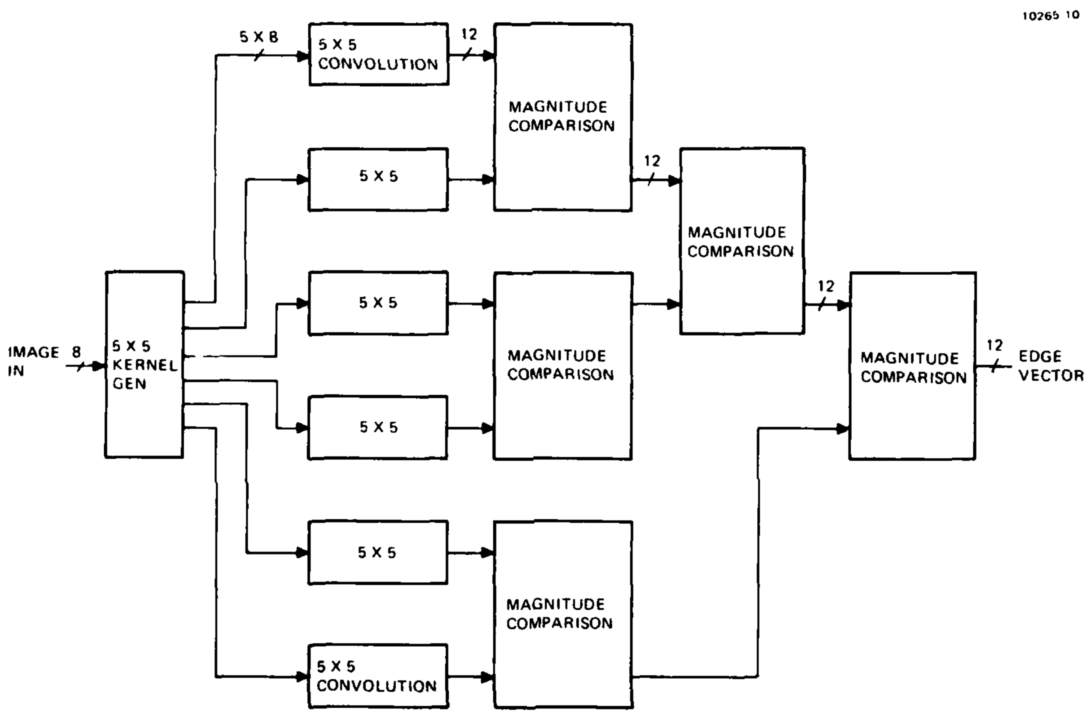


Figure 13. Logic diagram for edge detection.

10265-11

−100	−100	0	100	100
−100	−100	0	100	100
−100	−100	0	100	100
−100	−100	0	100	100
−100	−100	0	100	100

→

(a) 0°

−100	32	100	100	100
−100	−78	92	100	100
−100	−100	0	100	100
−100	−100	−92	78	100
−100	−100	−100	−32	100

↗

(b) 30°

100	100	100	100	100
−32	78	100	100	100
−100	−92	0	92	100
−100	−100	−100	−78	32
−100	−100	−100	−100	−100

↗

(c) 60°

100	100	100	100	100
100	100	100	100	100
0	0	0	0	0
−100	−100	−100	−100	−100
−100	−100	−100	−100	−100

↑

(d) 90°

100	100	100	100	100
100	100	100	78	−32
100	92	0	−92	−100
32	−78	−100	−100	−100
−100	−100	−100	−100	−100

↖

(e) 120°

100	100	100	32	−100
100	100	92	−78	−100
100	100	0	−100	−100
100	78	−92	−100	−100
100	32	−100	−100	−100

↖

(f) 150°

Figure 14. Edge masks $M_k'(i,j)$ in six directions.

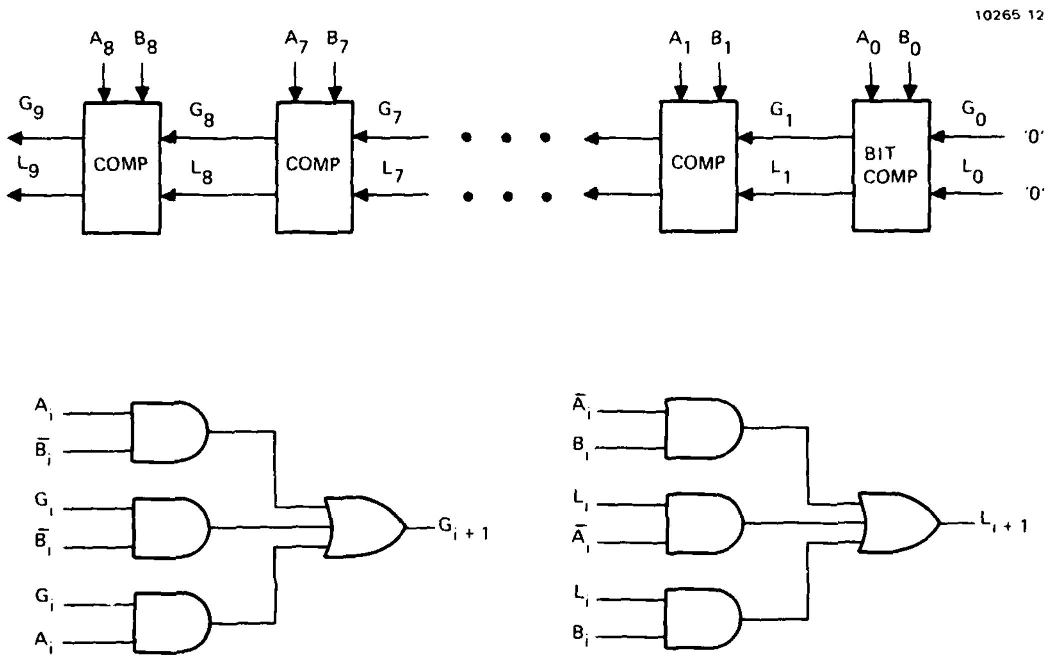
Figure 15 shows the logic for performing the magnitude comparison. The corresponding bits of each data word are compared in turn, and greater than (G) and less than (L) signals propagate towards the msb bits. G{9} and L{9} indicate whether A is greater than or less than B. If G{9} and L{9} are both low, then the two words are the same. Sixty four gates would be necessary to implement a full 8-bit comparator. Hence, 320 gates would be necessary to implement the five comparators in the edge detection step. It has been estimated that it takes 178K gates to implement the six residue convolvers. This is by far the most hardware intensive computational part.

B. THINNING AND THRESHOLDING

Thinning in Nevatia's process is accomplished by comparing the edge magnitude and direction at a pixel location with that of some of its surrounding eight neighbors. To qualify as an edge point, the following rules must be observed:

- The edge magnitude at the pixel location must be greater than that of its two neighbors in a direction normal to the direction of this edge. The normal to a 30° edge is approximated by the diagonals of a 3 x 3 grid.
- The edge directions of the two neighboring pixels, as defined above must be within 1 unit difference (30°) from that of the central pixel.
- The edge magnitude of the central pixel must exceed a certain fixed threshold. This threshold is arbitrarily set at some low value.

When implementing in hardware, the algorithm can be divided into two parts: the first accesses the edge magnitude and direction of the two neighbors normal to the direction of the edge; the second compares the magnitude and direction of the central pixel with those of the two neighbors and a fixed threshold to ensure that the second and third conditions above are met. Figure 16 shows the logic for performing thinning and thresholding. The eight neighbors of the central pixel are first sent to a switching network which decodes the edge direction of the central pixel and allows the edge data for the correct two neighbors to pass on to the comparison network. In the second



$G_{i+1} = A_i \bar{B}_i + G_i \bar{B}_i + G_i A_i$
(4 GATES)

$L_{i+1} = \bar{A}_i B_i + L_i \bar{A}_i + L_i B_i$
(4 GATES)

TOTAL NUMBER OF GATES TO IMPLEMENT 8 BIT COMPARATOR = 8 X (4 + 4) = 64 GATES

Figure 15. Logic for performing magnitude comparison.

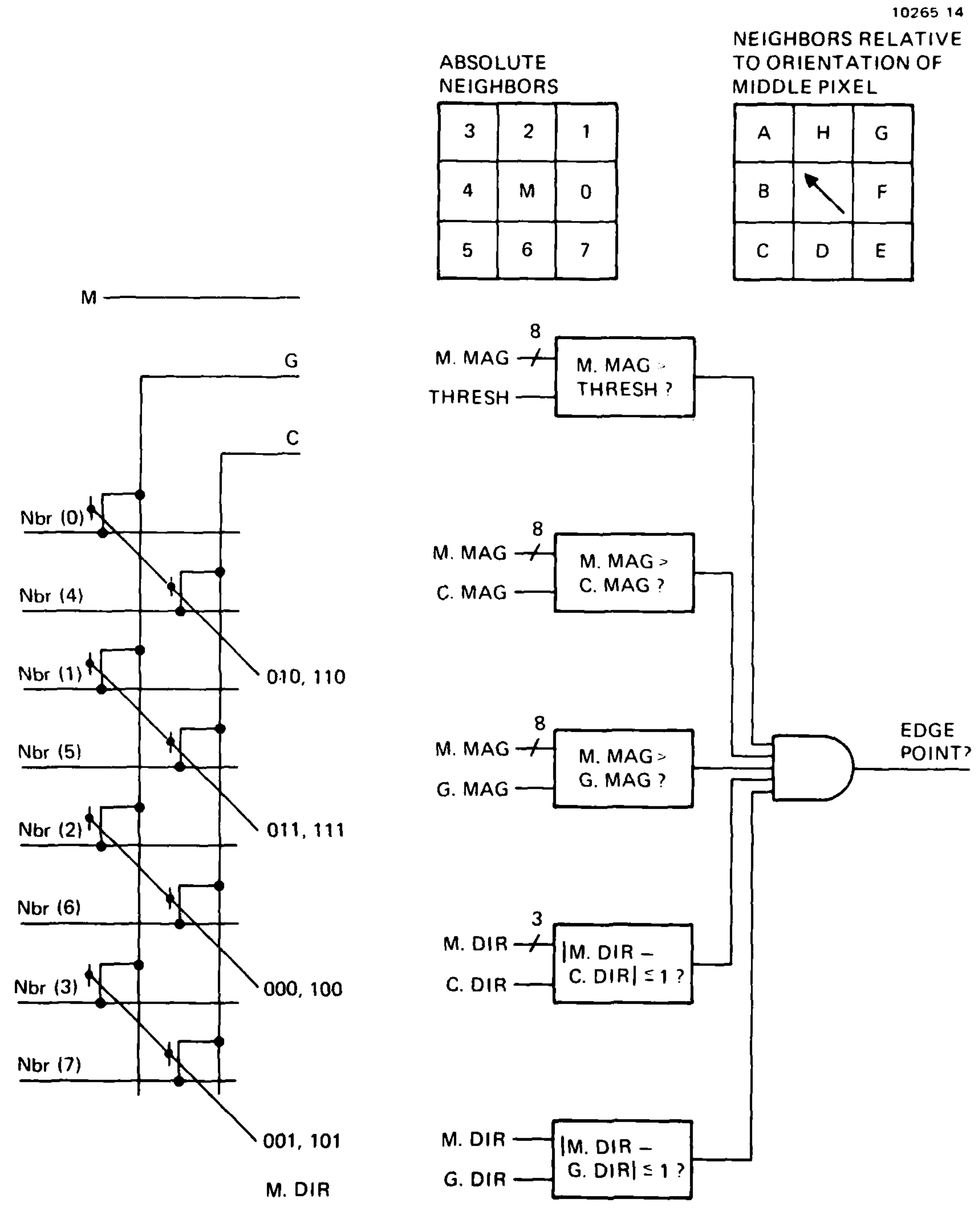


Figure 16. Logic for performing thinning and thresholding.

part, the edge magnitudes and directions are compared; and, if the second and third conditions are met, an edge point is presumed present. A gate count reveals that 190 gates are needed: 8 for the switching network and 181 for the comparison network.

C. EDGE LINKING

At this point, the data format for each pixel is pictured in Figure 17. One bit is assigned to indicate if the pixel is an edge point, 3 bits to indicate the direction of the edge, and 8 bits to indicate edge strength. The next step is to link the edge points together by forming predecessor and successor members. There are, at most, three possible candidates to be a predecessor or successor among the eight neighbors of an edge point. However, an edge point can have, at most, two predecessors and two successors. In such a case, only the primary predecessor (or successor) is encoded, and a special bit is marked to indicate the presence of a fork. The following rules are observed in edge linking:

- The orientation of a predecessor (successor) must be less than 1 unit difference (30°) from that of the central edge point.

- If there is more than 1 possible predecessor (successor), a fork will exist if they are not 4-neighbors, or if their orientations differ by 2 units (60°) if they are 4-neighbors. In such a case, the predecessor (successor) with the greater magnitude is encoded as the primary predecessor (successor). If the possible candidates are 4-neighbors with the same orientation, the chosen candidate is the nearer of the two in the Euclidean sense.

The interested reader is referred to Refs. 1 and 2 for more details on the rules concerning edge linking. Figure 18 shows some successor configurations illustrating the above rules.

The logic for edge linking is shown in Figure 19. In the first section, a shifting network decodes the edge orientation of the central pixel and accesses the edge data for the three possible successor (predecessor) locations. This edge data is then sent to a comparison network that tests whether the conditions above have been satisfied. The results are then sent to a PLA containing about 40 minterms, the output of which indicates the successor and whether a fork is present. The coding for the PLA is shown in Figure 20. The gate count is 6 for

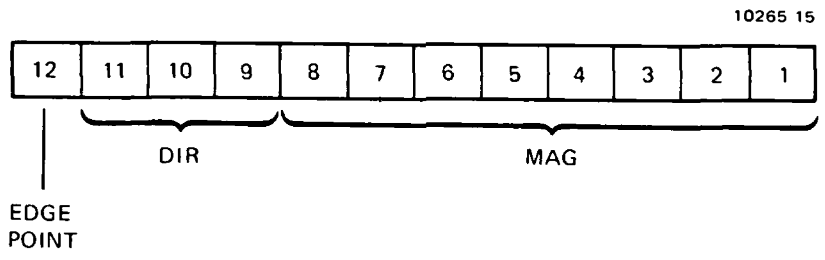


Figure 17. Data format for edge point after thinning and thresholding.

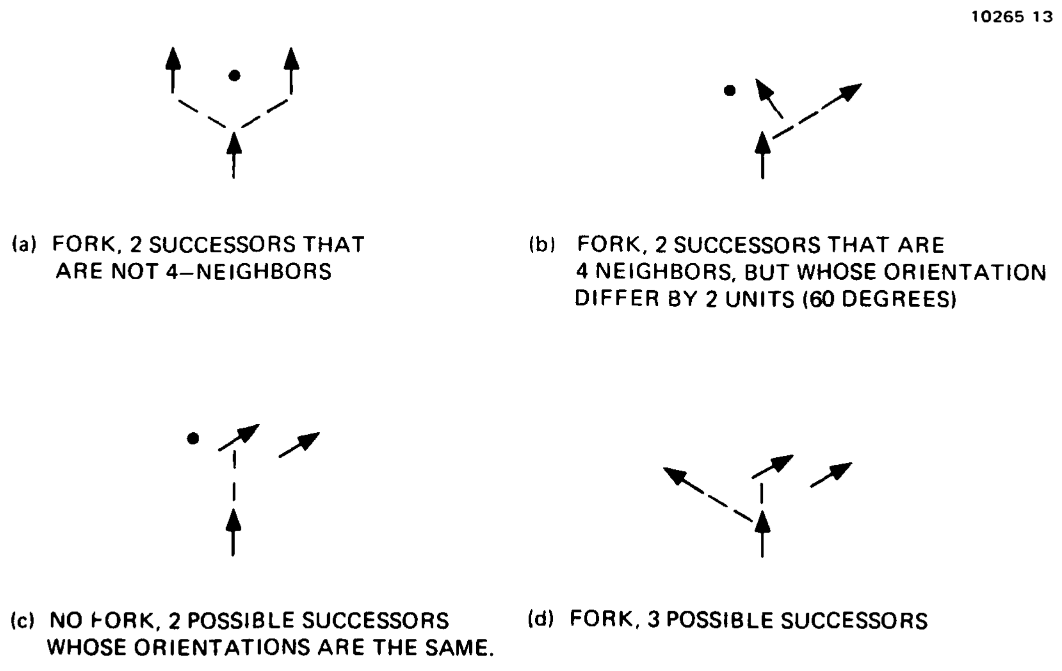

10265 13
(a) FORK, 2 SUCCESSORS THAT ARE NOT 4–NEIGHBORS
(b) FORK, 2 SUCCESSORS THAT ARE 4 NEIGHBORS, BUT WHOSE ORIENTATION DIFFER BY 2 UNITS (60 DEGREES)
(c) NO FORK, 2 POSSIBLE SUCCESSORS WHOSE ORIENTATIONS ARE THE SAME.
(d) FORK, 3 POSSIBLE SUCCESSORS

Figure 18. Some successor configurations illustrating edge linking rules.

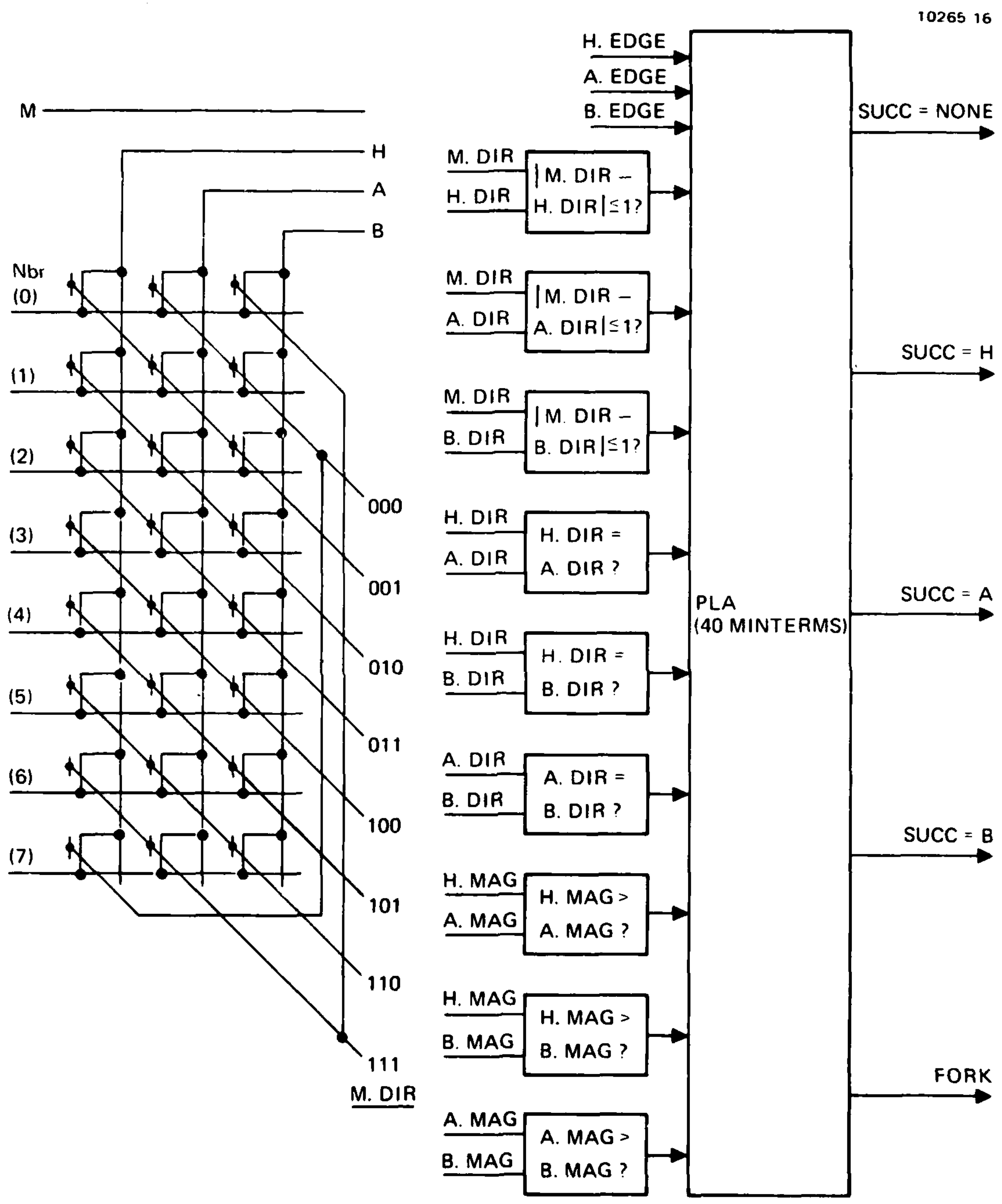


Figure 19. Logic for edge linking.

10265 17

H. EDGE	A EDGE	B. EDGE	M. DIR − H. DIR ≤ 1?	M. DIR − A. DIR ≤ 1?	M. DIR − B. DIR ≤ 1?	H. DIR = A. DIR ?	H. DIR = B. DIR ?	A. DIR = B. DIR ?	H. MAG > A. MAG ?	H. MAG > B. MAG ?	A. MAG > C. MAG ?	FORK	PRIMARY SUCC
0	0	0	−	−	−	−	−	−	−	−	−	0	0
0	0	1	−	−	0	−	−	−	−	−	−	0	0
			−	−	1	−	−	−	−	−	−	0	B
0	1	0	−	0	−	−	−	−	−	−	−	0	0
			−	1	−							0	A
1	0	0	0	−	−							0	0
			1	−	−							0	H
0	1	1	−	0	0							0	0
			−	0	1							0	B
			−	1	0							0	A
			−	1	1	−	−	0	−	−	0	1	B
									−	−	1	1	A
						−	−	1	−	−	−	0	A
1	0	1	0	−	0							0	0
			0	−	1							0	B
			1	−	0							0	H
			1	−	1	−	−	−	−	0	−	1	B
									−	1	−	1	H
1	1	0	0	0	−							0	0
			0	1	−							0	A
			1	0	−							0	H
			1	1	−	0	−	−	0	−	−	1	A
									1	−	−	1	H
						1	−	−	−	−	−	0	A
1	1	1	0	0	0							0	0
			1	0	0							0	H
			0	1	0							0	A
			0	0	1							0	B
			1	0	1	−	−	−	−	0	−	1	B
									−	1	−	1	H
			1	1	0	0	−	−	0	−	−	1	A
									1	−	−	1	H
						1	−	−	−	−	−	0	A
			0	1	1	−	−	0	−	−	0	1	B
									−	−	1	1	A
						−	−	1	−	−	−	0	A
			1	1	1	1	0	0	−	−	0	1	B
									−	−	1	1	A
						0	0	1	0	−	−	1	A
									1	−	−	1	H

Figure 20. Coding for edge linking PLA.

the shifting network, 72 for the comparison network, and 170 for the PLA, for a total of about 250 gates. Formation of the predecessor numbers would take another similar sized network, bringing the total number of gates to about 500.

D. EDGE TRACING

In this step, the predecessor/successor (PS) elements formed in the previous operation are linked together. The data format for the PS file is shown in Figure 21. Included in this format is a trace bit which indicates whether the point has already been collected. The PS elements are linked in three passes:

- In the first pass, a raster scan is made of the PS files, in search of an edge point with no predecessor. Edge tracing begins at these points, and only the primary successor elements are linked when there is a fork.
- In the second pass, the edge points that have forks are revisited, and the secondary successor elements are linked.
- In the third pass, circular edge segments with no starting or fork points are linked. This requires scanning the trace file to find an edge point that has not been collected before, and then tracing out the circular segment.

It would be undesirable to perform edge tracing in three passes if the operation is to be done in real time, since the requirements on buffering and speed of the hardware would then be quite severe. The three passes for the operation could be reduced if all the start and fork points were identified in a previous step, and the addresses stored in a list. The start address for tracing an edge segment could then be provided by this list instead of by scanning the PS file. This could result in considerable time saving, since typically less than 10 percent of an image are edge points, and only a fraction of those are start and fork points.

The logic for edge tracing is shown in Figure 22. Two 5.5 megabit memories are needed for storing the PS files. While edge tracing is being performed on the edge data in one memory, the other memory is being filled with fresh data. The "start and fork" list provides the starting address of an edge segment. The PS information fetched for this point is used to form the edgr linked list, and also to generate the address of the successor to be fetched. The address

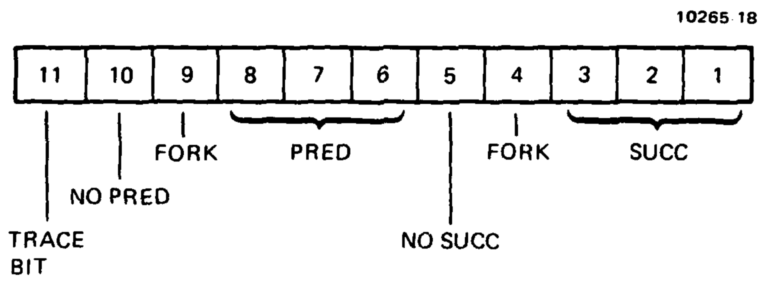


Figure 21. Predecessor/successor data format.

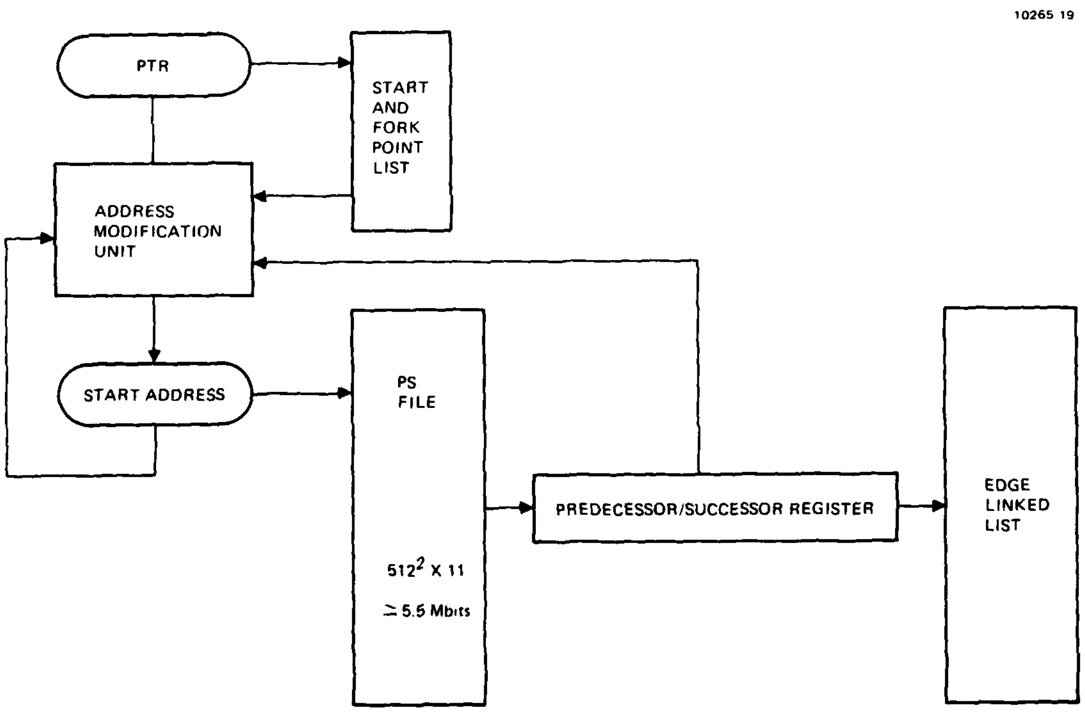

10265 19
PTR
START AND FORK POINT LIST
ADDRESS MODIFICATION UNIT
START ADDRESS
PS FILE
512² X 11
≃ 5.5 Mbits
PREDECESSOR/SUCCESSOR REGISTER
EDGE LINKED LIST

Figure 22. Logic for edge tracing.

modification unit acts as a controller enforcing the three rules above and may broadcast the address of a new pixel to be fetched, modify the address of the previous pixel fetched, or decide that a fork is present and broadcast the address of the secondary successor that is to be fetched.

E. GATE COUNT TABULATION

Table 2 summarizes the results of the line finder system, presenting the number of gates required for each function. As with the texture system, these data will be used later in the study for VLSI partitioning.

SECTION 6

SYSTEM INTEGRATION TO HOST PROCESSOR

A principal area of our present work has been to analyze and develop hardware that will be integrated into a commercially available test system to make it portable and extendable. One factor expected to affect the partitioning substantially is the input/output requirements of the system. The data flow graphs specify the bus widths and the basic data inputs and outputs, bu ute bandwidths and control requirements cannot be specified until an operati environment has been defined. In the past, when we have fabricated CCD LSI image processing chips, they have been used in a stand-alone system with dedicated input and output devices. The bandwidths of these devices specified the bandwidth of the data paths on the chip. But we do not expect some of the systems being examined in this study to be used in a stand alone system; instead, we expect them to be used as a peripheral device to a host general-purpose processor. This then requires that we expand our system study to include the implications of using a general purpose computer to host the IU systems.

REFERENCES

1. R. Nevatia and K.R. Babu, "Linear Feature Extraction and Description," published in CUIP, 1980.

2. R. Nevatia and K.R. Babu, "An Edge Detection, Linking and Line Finding Program," USC IPI Report No. 840, Sept. 1978.

3. C.S. Swigert, "Decoding the Edge Detection Linking and Line Finding Program of R. Nevatia and R. Babu," HAC Memo ESD 545, Oct. 1979.

4. V.S. Wong, "Description of Edge Finding Process in Thesis by V.S. Wong," HAC Memo ESD 303, Aug. 1980.

5. S.D. Fouse, "Proposal for Residue Convolver Chip," HAC Memo ESD 242, July 1980.

6. R. Babu, Private communication.

7. R.O. Duda and P.E. Hart Pattern Classification and Scene Analysis, (Wiley,]973, 338-339).

PROGRESS REPORT

October 1980 to April 1981

SECTION 1

INTRODUCTION

Our previous work[1,2] in developing image understanding architectures has concentrated on the analysis of the processing functions required for special purpose LSI primitives. We have developed about 16 fixed and programmable primitives for real-time operation.

The work described here represents a significant shift in emphasis and an increase in capability. First, we have undertaken a detailed design and analysis of a number of complex processing operations, including line-finding[3] and texture analysis[4]. This work has been carried out specifically with LSI and VLSI implementation in mind. Hence, issues such as chip and function partitioning, data flow, local storage, and wordlength are specifically emphasized. The results of the systems analysis and design for these operations are included in Section 2. From this work we have been able to configure a fully integrated real-time processor for each.

Of equal importance, and perhaps greater impact to military systems and robotics, we have configured, designed, and started to fabricate a VLSI processor that can form the basis of a fully programmable image understanding system compatible with commercially available host machines. The architecture itself uses residue arithmetic[5] to provide a highly regular and extendable structure. These issues are of great importance in the emerging VLSI era where design time and the ability to amortize the fabrication cost of many processors are essential elements. The VLSI processor now under development is configured on a single board with multiple copies of a single custom-built nMOS chip. Our estimates indicate that the processor will perform between 50 and 75 percent of the operations for line finding and texture classification. The modular nature of the machine can provide essentially variable precision, as discussed in Section 3. The single custom-built chip has a complexity equivalent to approximately 6,500 transistors. However, with a decrease in design rules from the present 5 μm to submicron we can anticipate building a single chip with some 80,000 transistors and design the whole system around four identical chips.

A significant advantage of our approach is the compatibility with general purpose host machines, such as the DEC series, which are widely used in image analysis and understanding. We have therefore spent considerable effort in developing a UNIBUS interface so that the machine can be accessed through the host software. With the addition of the local area logic processor, to be developed in the next phase, we expect to demonstrate a fully programmable real-time processor.

SECTION 2

SYSTEMS ANALYSIS AND DESIGN

The effective exploitation of VLSI technology in image understanding systems requires that the processors developed be used in as wide a range of systems as possible. This requires that a wide variety of systems be analyzed for the purpose of determining commonality. Our approach has been to select three representative systems to analyze: a line finder, a texture analyzer, and a segmentor.[6] Each system was studied and then a directed graph depicting the data flow was produced.[7] The directed graphs had nodes that were functionally complex, so the next step was to perform a logic design for the systems to determine the complexity of the nodes and of the systems. The logic design was done for the line-finder and the texture analyzer systems. For details of each design, see USCIPI Report No. 990.[2] A brief summary of the results for each system are presented below.

The line-finder system breaks down into four major functions: edge detection, edge thinning, edge linking, and edge tracing.

A design was generated for each of these functions and Table 1 presents the number of gates each required. Similarly, the texture analysis system divides into five major functions: small window convolution (5 x 5), small window statistical calculation, scaling, large window statistical calculation, linear transformation; Table 2 presents the gate count for each system.

From the results of the directed graph analysis and the logic design, it is obvious that the function that is common to all three systems, and the most complex when measured by the number of gates, is the small window (5 x 5) convolution. This supports our decision to build a programmable 5 x 5 local-area processor as the basic VLSI module.

Table 1. Gate Count for Line Finder System

Edge Detection	178K
Thinning	190 Gates
Edge Linking	500
Edge Tracing	12 Mbit Memory +5K Logic Gates

Table 2. Gate Count for Texture Classification System

5 Line Kernel Generation	15K Gates
5 x 5 Convolution	27K Gates/Channel
5 x 5 Variance	10K Gates
Normalization	1K/Channel
Large Window Statistical Calculation	8.25K/Channel
Transform (M Input Channels)	2.1K·M/Output Channel

SECTION 3

A RESIDUE BASED IMAGE PROCESSOR

The work described in Section 2 motivated the design of a processor which could perform the computationally intensive low-level operations for each of the three systems investigated. In addition to fulfilling the requirements of the three systems, we also wanted to select an architecture which could be extended to take advantage of the VLSI design and processing capabilities which are currently being developed. The architecture we selected is based on the technique of residue arithmetic.

A. PROCESSOR DESCRIPTION

We implemented our local area processor in residue arithmetic to take advantage of modularity, and hence ease of design, within the VLSI chip, and extendability to handle arbitrary dynamic range and accuracy. The technique relies on the conversion, prior to computation, of all the data to relatively prime bases (we chose 31, 29, 23, and 19) and the subsequent decoding of the processed data back to binary numbers. If this overhead is accepted then the arithmetic itself is reduced, both in complexity and in required dynamic range. This enables us to use look-up tables, which in our case are programmable RAMs, to perform the necessary arithmetic. Regularity, ease of VLSI design, and function density are significant advantages. Thus, this approach is ideal for VLSI implementation.

A block diagram of a general residue processor is shown in Figure 1. Some of the advantages (e.g., modularity) and disadvantages (encoding, etc.) of this technique are clearly visible in this representation. The encoding and decoding, when compared to a binary processor, are overhead functions and can be the major disadvantage of a residue processor. However, this overhead cost can be reduced if enough computations can be performed while in the residue representation, and hence the encoding and decoding can be amortized over a large computation base. The clear advantage of this type of processor is its natural parallelism. Each parallel computation channel is independent, requiring no communication with its neighbors until the conversion from the residue representation to a binary representation is performed.

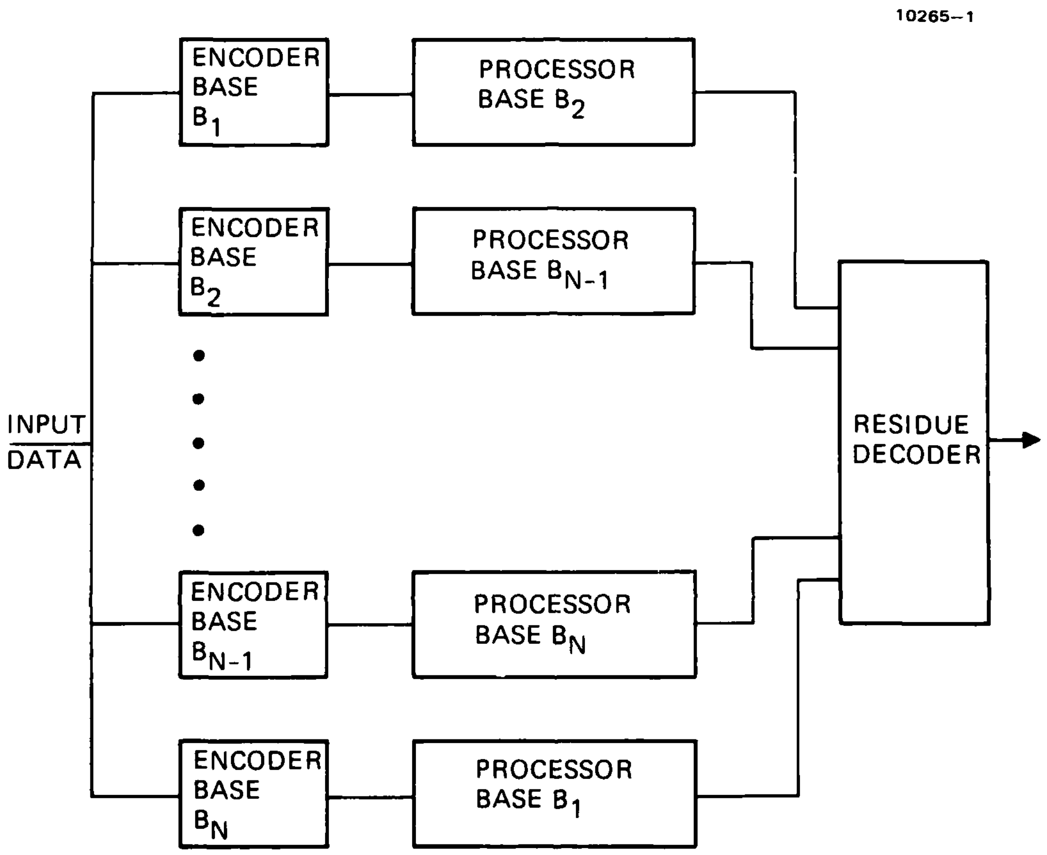


Figure 1. General structure for residue processor.

B. KERNEL GENERATION AND ENCODING

Typically, the input to an image processor is a string of 8-bit data values generated by a raster scan of the image, in which case we must include in the processor the means for generating the two-dimensional kernel. This kernel generation function is most easily accomplished using a series of shift registers. For a five line kernel, four shift registers, each one containing as many elements as there are pixels in a line, are required to generate five adjacent lines of video. For our particular application the shift registers are 8 bits wide and 512 elements long.

Before the input data can be processed by a residue processor it must be converted from a binary representation to a residue representation. This conversion requires that we calculate

$$(X \bmod B1,\ X \bmod B2,\ \ldots,\ X \bmod Bn),$$

where X is the value of the input data and Bi is the ith base. For our case, since we operate on a 5 x 5 kernel, we must perform this calculation on the input for each of 5 lines of video and for each of 4 processors (equivalent to the four bases). The simplest way to perform this calculation for a general set of bases is to use Read Only Memories (ROMs). By connecting the input data to the address lines of the ROM and looking at the ROM's data lines for the output, a look-up function is performed. For our particular processor, which will support an 8-bit input dynamic range and bases which can be encoded in 5 bits or less, the size of an encoding ROM is 256 x 5 bits. Figure 2 shows the block diagram for the kernel generation and encoding portion of a 4 base 5 line processor. Each of the 5 ROMs for each base are programmed identically.

An alternative way to perform these two functions would be to encode before the kernel is generated. The major drawback of this technique is that the memory requirements are much greater for the kernel generation process. For the system we are currently constructing, each line delay would need to be 20 bits wide, as opposed to 8 bits wide for the method we chose.

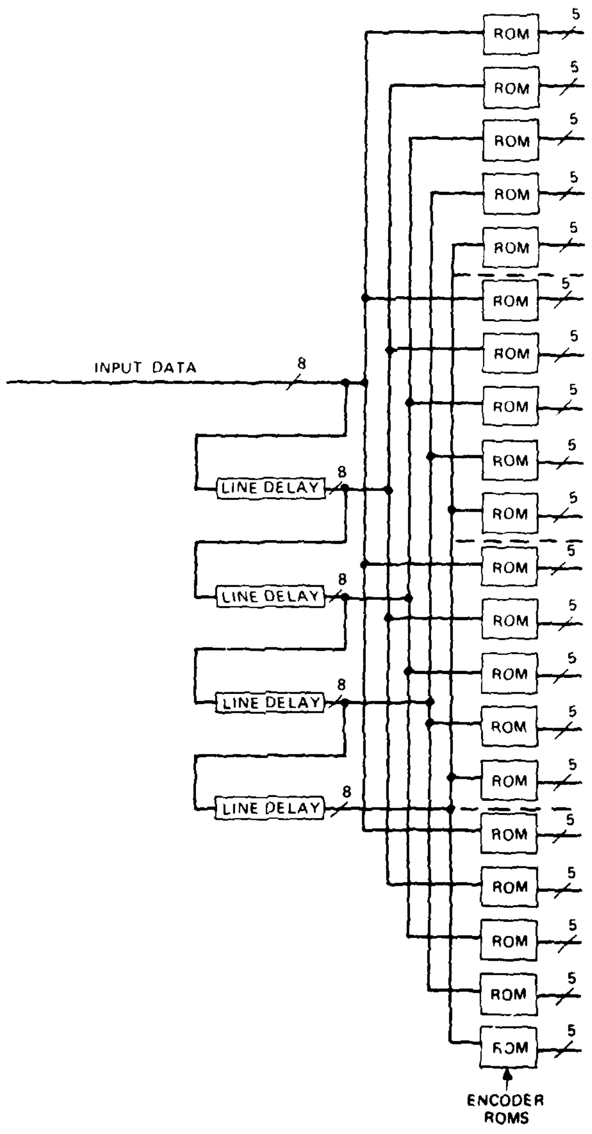


Figure 2. Kernel generator encoding for 5x5 processor.

C. A PROGRAMMABLE RESIDUE COMPUTATION LSI CIRCUIT

The actual computations on the image data will be performed by a custom LSI circuit which is currently being processed at the Hughes Carlsbad Research Center. The circuit will process a 5 x 1 kernel and is capable of performing computations of the form

$$y = f_i(x_i),$$

where y is the output value, x_i are the five elements in the kernel, and f_i represents polynomial functions of a single variable.

A functional block diagram of the circuit is shown in Figure 3. The word size for this is 5 bits, which limits the prime bases used to a value of 32 or less. The circuit is designed to accept a 5-bit input word which is clocked into a 5 element shift register. The contents of each register element is then shifted to the next register. The 5-bit data in each of the shift register elements is used to address a look-up table, which is a 32 x 5 Random Access Memory (RAM). This look-up operation performs a unary operation, such as a multiplication by a constant or a squaring operation. The outputs of the 5 RAMS are then summed modularly to produce a 5-bit output, the base of the modular addition being programmable by external control of the circuit. Since the look-up tables which perform the unary operation are composed of RAMs, the circuit can be programmed for many different computations, such as different weights for a convolution or different powers of a number for a statistical calculation.

A detailed schematic of the circuit is shown in Figure 4. In addition to having 5 bits of input data and 5 bits of output data, an additional set of data lines is included in this design. These data lines, which are bi-directional, serve a multipurpose role for control and testing. When used as input data lines they can be used to program the base of the modular addition and to program any of the 5 look-up tables. When used as output data lines they can read the look-up tables to verify the operation of the circuit.

This circuit is being fabricated using the nMOS technology and has been designed to accept a 10 MHz data rate. To achieve this data rate pipeline techniques were used; the resulting latency for this circuit is 7 clock cycles. The circuit will be packaged in a 28-pin DIP. Figure 5 is a photograph of the layout of the chip.

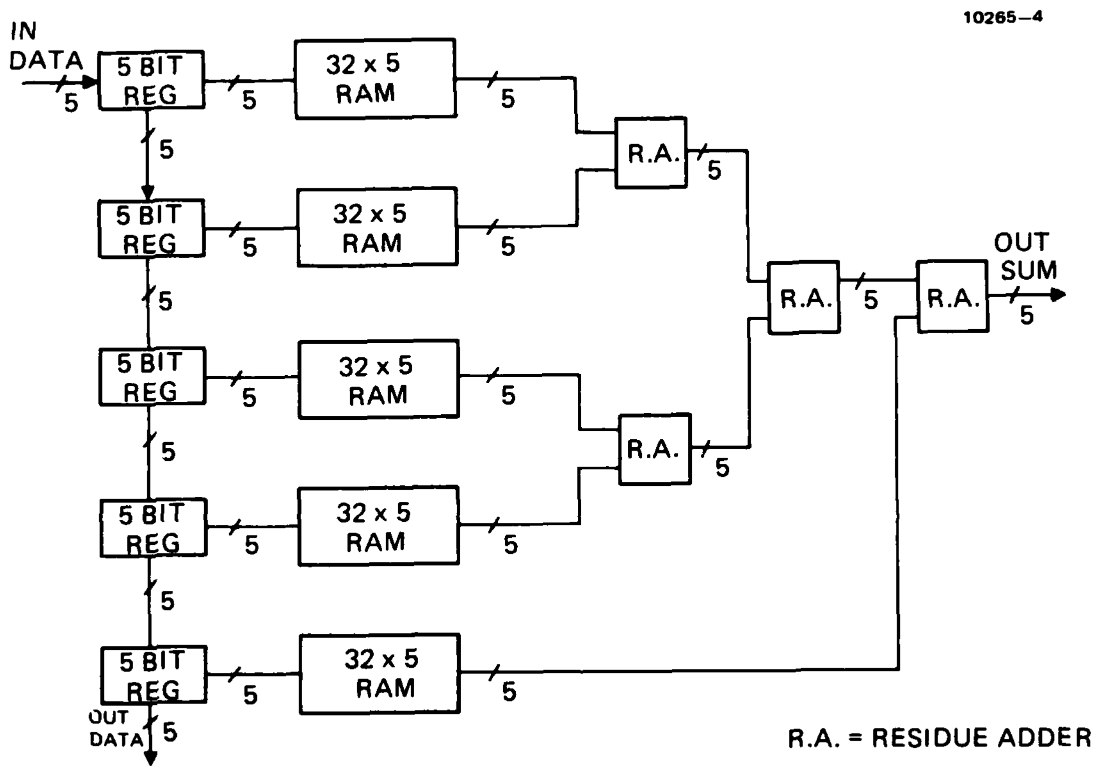


Figure 3. A functional block diagram for 5x1 residue processor circuit.

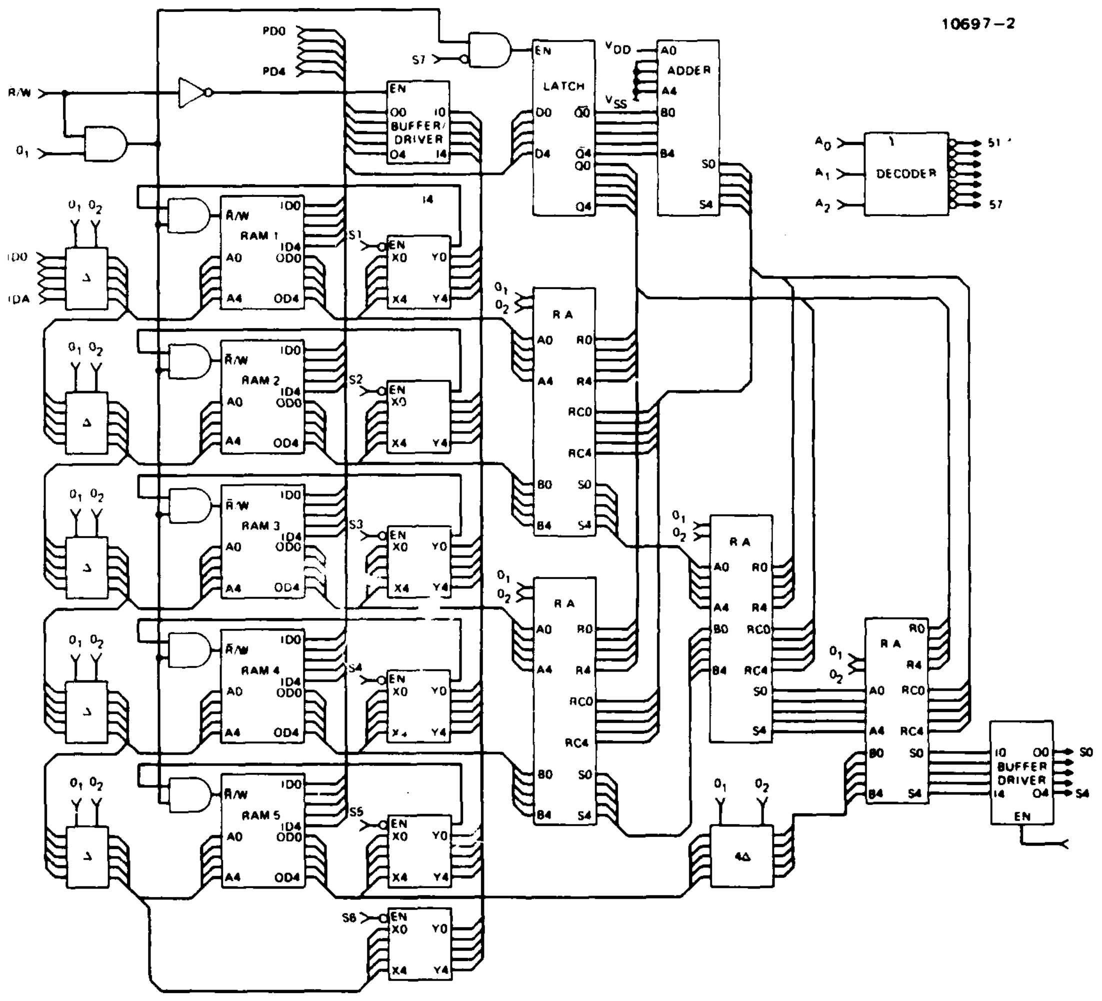


Figure 4. Schematic of 5x1 residue circuit

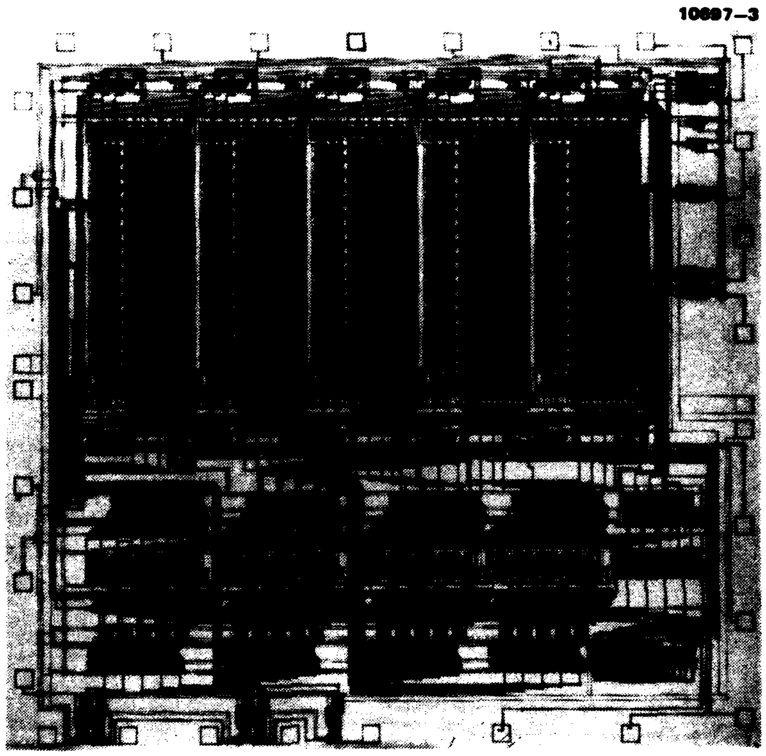


Figure 5. Photograph of CRC 181 layout.

To utilize this circuit (with a 5 x 1 kernel) in a 5 x 5 local area processor, multiple copies of the circuit are used, as well as additional logic to combine the outputs of the individual circuits. For each base, 5 of these circuits are used, one for each line of the kernel. In addition, four 1,024 x 5-bit ROMs are used to sum the outputs of the 5 circuits. ROMs are used instead of adders because the additions must be done modularly. Figure 6 shows the block diagram of the processor, including the encoding and computation portions.

D. DECODING

The last portion of the processor is concerned with the conversion from the residue representation to a binary representation. This conversion could certainly be done the same way as the encoding (by table look-up), but there is a severe problem with that approach. For our particular system, to convert four 5-bit values, the decoder would require a memory 1 million elements wide, with each element being 17 bits deep. This table is certainly attainable, but the approach is not extendable. If an extra base is required so that five 5-bit values need to be converted, the memory requirements increase to 33 million elements, each being greater than 20 bits deep.

There are two conversion methods that do not require these large memories. One is based on the Chinese Remainder Theorem and the other is based on a mixed radix representation. (For a complete discussion, refer to Ref. 9). This report will focus only on the particular implementations of these techniques and the rationale for selecting one over the other.

To be able to reasonably discuss either of the two conversion methods some notation must be introduced. If B is the base vector whose elements are the bases used for the computations, or

$$B = (b_1, b_2, \ldots, b_k),$$

and R is a scalar whose value is equal to the dynamic range of the processor, which is given by

$$R = b_1 * b_2 * \ldots * b_k,$$

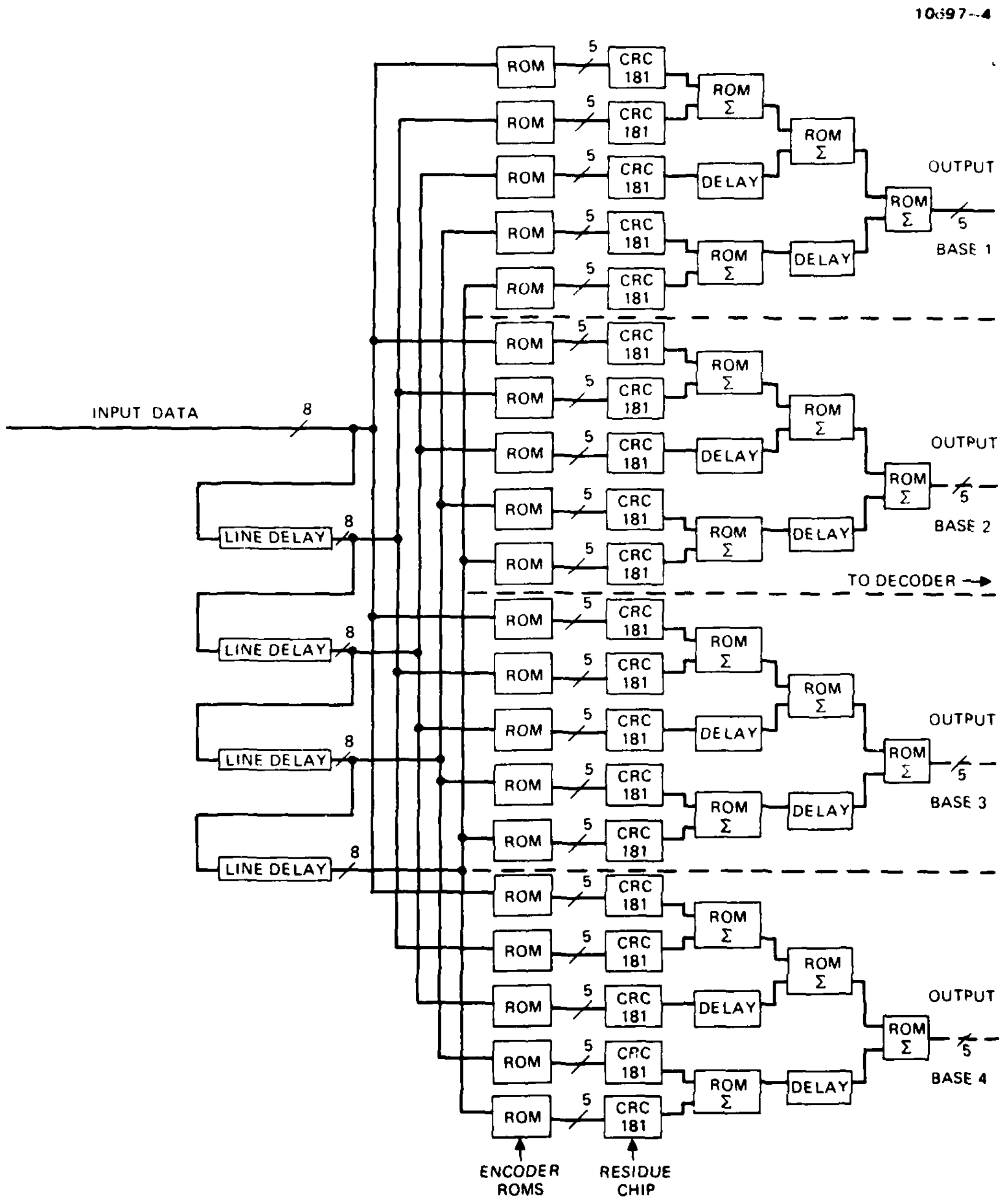


Figure 6. Structure of 5x5 processor utilizing 5x1 processor circuits.

and x is the value we wish to encode into the residue representation, then RX, the vector whose elements are the data values for each of the computation channels, is given by

$$RX = (rx_1, rx_2, \ldots, rx_k),$$

where

$$rx_i = x \text{ MOD } b_i, \ i = 1 \text{ to } k.$$

The Chinese Remainder conversion process is based on the following property. If

$$RS = (rx_1, rx_2, rx_3, rx_4),$$

then

$$RX = [(rx_1, rx_2, 0, 0) + (0, 0, rx_3, rx_4)] \text{ MOD } r.$$

Figure 7 shows a system which performs this conversion and which requires only two blocks of memory 1,024 elements wide with two adders. The adders need only be as large as the accuracy required of the system. Typically, for image processing systems, the output dynamic range and the input dyanamic range are equal, and thus the adder complexity can be relatively small.

The second conversion scheme considered is based on the mixed radix method, but is simplified by the fact that the output dynamic range can be approximately 8 bits. The method can be explained by imagining an iterative process where at every iteration the smallest base is eliminated by dividing the data value by that base value. Dividing essentially reduces the dynamic range of the value and thus eliminates the need for the extra base. Of course, since we are limited to a strictly integer system, we must make sure that the value is evenly divisible by the smallest base. This can be done by rounding up or down so that the element in the residue vector for that base is zero. Figure 8 shows an architecture for a 4 base system that performs this mixed radix-like conversion.

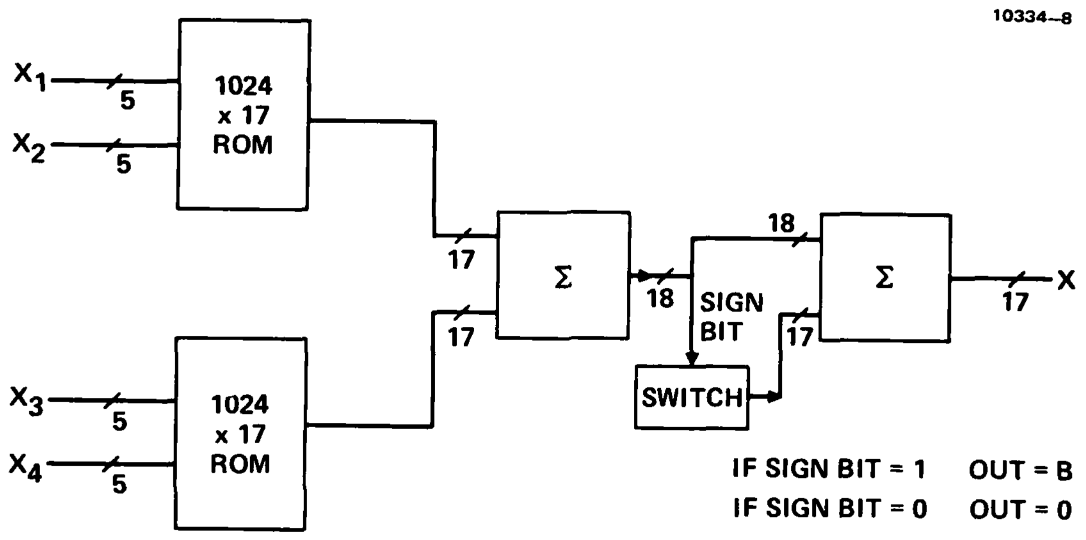


Figure 7. Chinese remainder theorem residue decoder (4 base system, 5 bits/base).

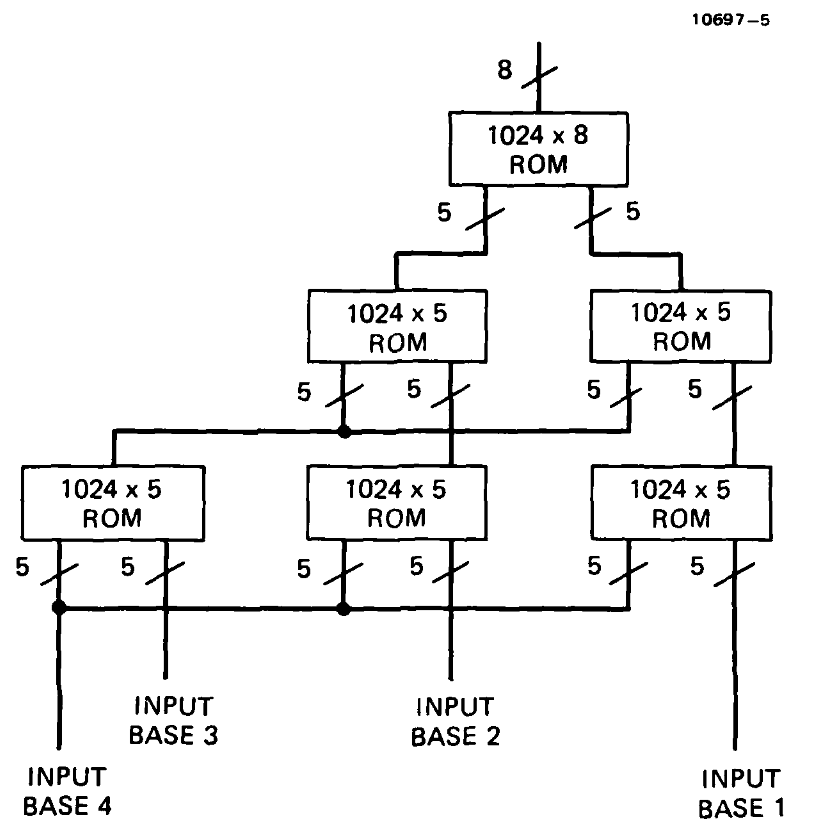


Figure 8. Mixed radix based residue decoder (4 base system, 5 bits/base).

At the bottom level of this tree structure the fourth base is eliminated. At the next highest level of the tree the third base is eliminated. Finally, we are left with two base values which can be decoded with a simple look-up table. This system has been simulated and the computer programs exist which can generate the contents of the ROMs for this conversion process for an arbitrary set of bases.

We chose the mixed radix-based conversion process to be implemented for our processor for two reasons. First, the method does not require any logic other than ROMs. This tends to make it more flexible and reliable. Second, the method appears to be easily extended to more bases by simply extending the decoding tree. Extending the Chinese Remainder based process would require either larger ROMs or more adders, either way being a less attractive method than the mixed radix type conversion.

E. PROGRAMMING AND CONTROL

A major problem in the fabrication of this processor is gaining access to each of the 20 custom residue chips for the purpose of programming the look-up tables. Each chip has three address lines to select one of 5 RAM structures, a read/write line, the 5 bidirectional programming data lines, and a control line for the data line drivers. These 10 control lines must be brought out for each of the 20 custom chips for a total of 200 control lines for the purpose of programming. However, by bussing lines where possible and by using a peripheral interface chip, the Intel 8255, the number of lines that are actually brought out of the processor is reduced to 16.

Figure 9 shows the structure that will be used to program the processor. The three address lines and the bus driver control lines are brought from each custom chip to an 8255. One 8255 is able to control 5 of the custom chips, since the 8255 has 24 lines available through three 8-bit ports. Thus, four 8255s are required to control all 20 of the custom chips. In addition, the 5 program data lines and the read/write line are bussed between each of the chips and these 6 lines are brought to a fifth 8255. The 8 input data lines of the 8255s are bussed together, as well as the 2-bit port select address lines. Finally, we bring out each of the 5 chip select lines to a binary decoder, allowing selection of a single 8255 using three control lines.

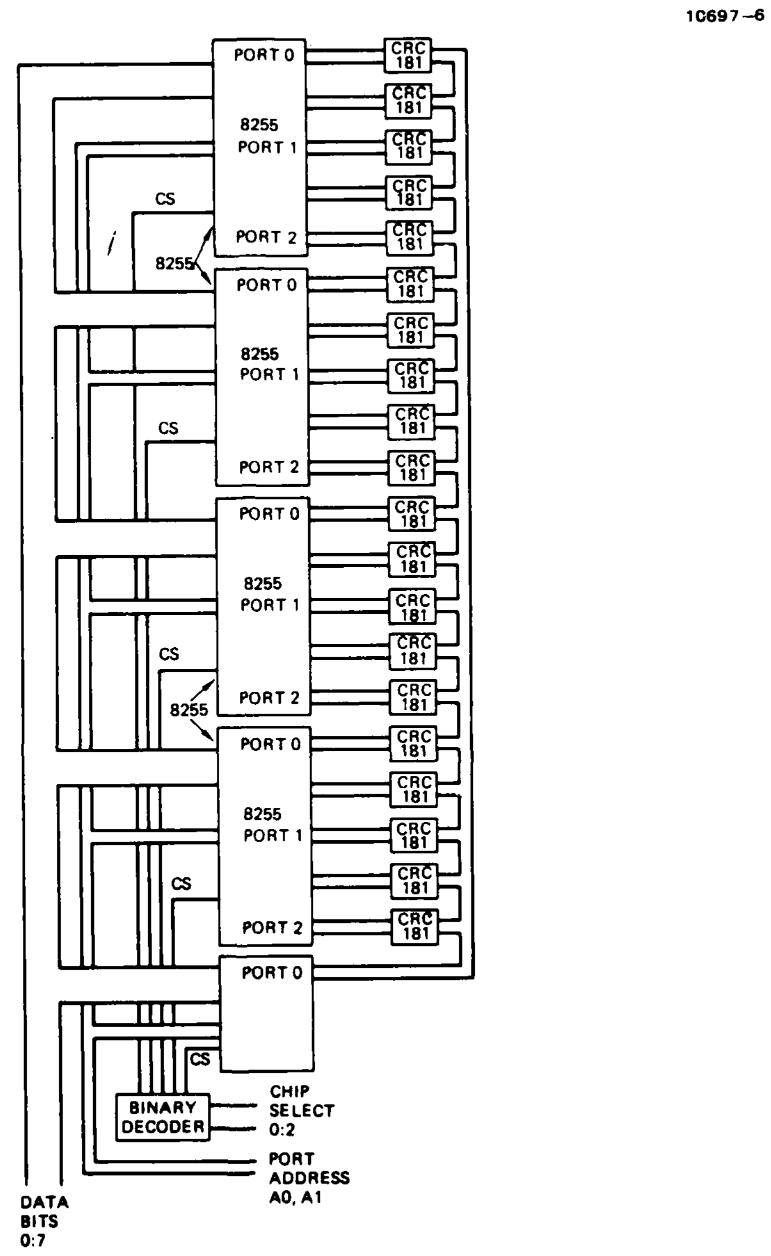


Figure 9. Structure for programming and control of residue processor.

To use this structure to program a given RAM element in the processor requires the following steps. Initially, the fifth 8255 is selected and the data to be programmed are written to the port containing the 5 program data lines. Next, the 8255 that controls the chip that the desired RAM element is on is selected, and the code to select the desired RAM structure is written to the proper port. Next, the address of the desired RAM element is provided on the input data lines of the processor, and then the address data is shifted so that it is addressing the proper RAM structure. Finally, the write data line is strobed to complete the programming sequence. This sequence can be accomplished by three 16-bit data transfers. For a processor using 31, 29, 23, and 19 as the modular bases, a total of 2,550 RAM elements need to be programmed. Thus, if three word transfers are required for each RAM element, a total of 7,650 word transfers are required to completely program the processor.

The processor, which involves all of the functions described above, is currently being fabricated. The majority of the electronics will be on a single wirewrap board, but the line delays will be in a separate box. A picture illustrating the progress on the wiring of the board is shown in Figure 10.

F. A VLSI VERSION OF A RESIDUE COMPUTATION CHIP

Even though the custom chip is performing the bulk of the computations, there is still a fair amount of extra circuitry required. Most of the extra circuitry is necessary because we are using a custom circuit based on a 5 x 1 kernel. The next step then, once this processor is tested and demonstrated, is to develop a 5 x 5 residue custom circuit and fabricate a processor which would utilize these VLSI circuits.

Ideally, the 5 x 5 circuit should include the circuitry for generating the 5-line kernel. This would mean that there would be 5 bits for input and 5 bits for output. If the kernel generation is performed off of the chip, 25 lines for input would be required, or at least a high speed multiplexer would need to be on the chip. On the other hand, if a simple line delay is used to generate the f-line kernel, the circuit would be too inflexible, since it may not be suited to certain applications. This can be avoided by augmenting the shift register with logic to control the clocking. By multiple clocking, the shift registers can be made to delay any length up to the maximum length. In other words, we would construct an elastic delay line.

10697-2

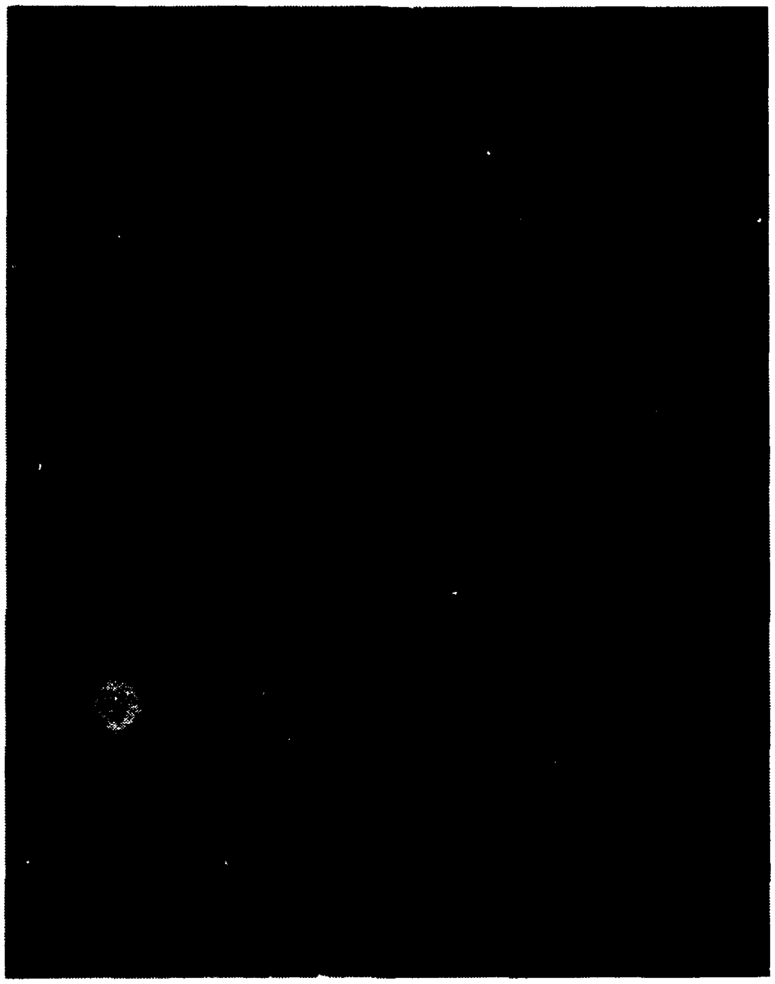


Figure 10. Photograph of processor wirewrap board.

Figure 11 shows a block diagram for the data flow of a 5 x 5 residue circuit. This circuit includes four delay lines, probably 512 elements long, 25 registers for generating the 5 x 5 window, 25 RAM structures, 24 modular adders, and some delay stages. In addition, programming, control, and testing of this circuit must be considered. A great deal can be learned about these issues by using the processor currently being fabricated.

Both the current chip, the 5 x 1, and the next chip, the 5 x 5, have been sized to get a quantitative measure of their complexity. The CRC 181 has a device count of approximately 6,500, of which the RAM portions of the circuit take up 4,500 devices. For the 5 x 5 custom circuit the device count will increase to 80,000. One of the reasons for the high device count is the addition of the line delays, which account for 50,000 devices (for static memory cells). The total number of devices for random logic is about 7,000, which is low considering that the circuit will have a throughput of 500 million operations per second.

As stated, going to a 5 x 5 circuit will greatly reduce the extra circuitry required to construct a 5 x 5 processor. Figure 12 shows a block diagram of a system utilizing a 5 x 5 circuit. With the VLSI circuit, the package count for the data flow portion of the processor will be only 14. This is compared to a package count in excess of one hundred for the current design. The power and size will be greatly reduced, thereby permitting the processor and the DEC UNIBUS interface to be put on a single card.

G. FUNCTIONAL CAPABILITIES

Although the primary motivation for developing this processor was a system that would require 5 x 5 convolutions, the processor is capable of performing a wider range of computations. The reason for this flexibility is due to the fact that we used a look-up table to perform a unary operation and that the table is completely programmable. The general form of the computation that can be performed by the processor is

$$y = f_i(x_i),$$

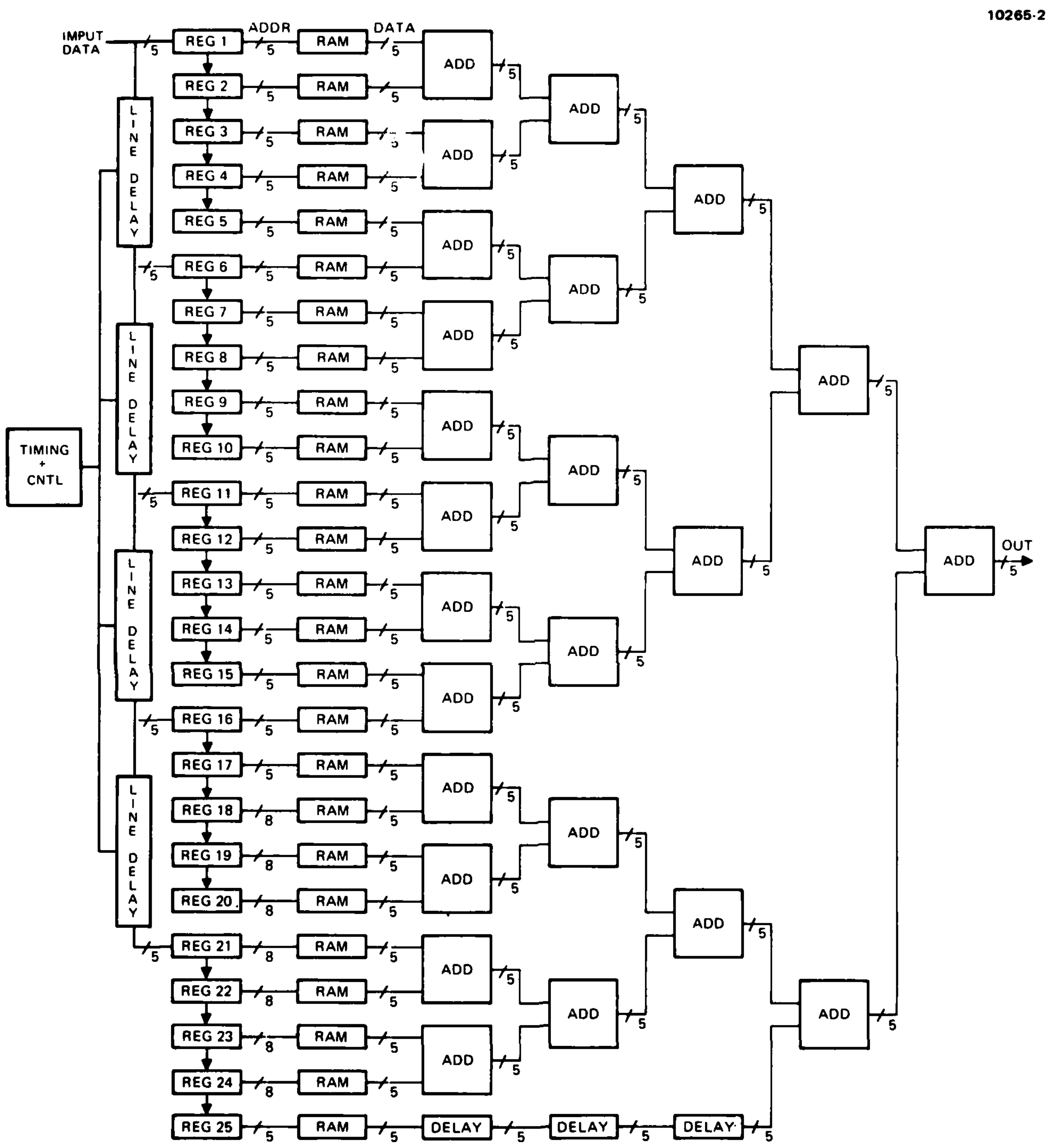


Figure 11. Data flow for 5x5 residue custom circuit.

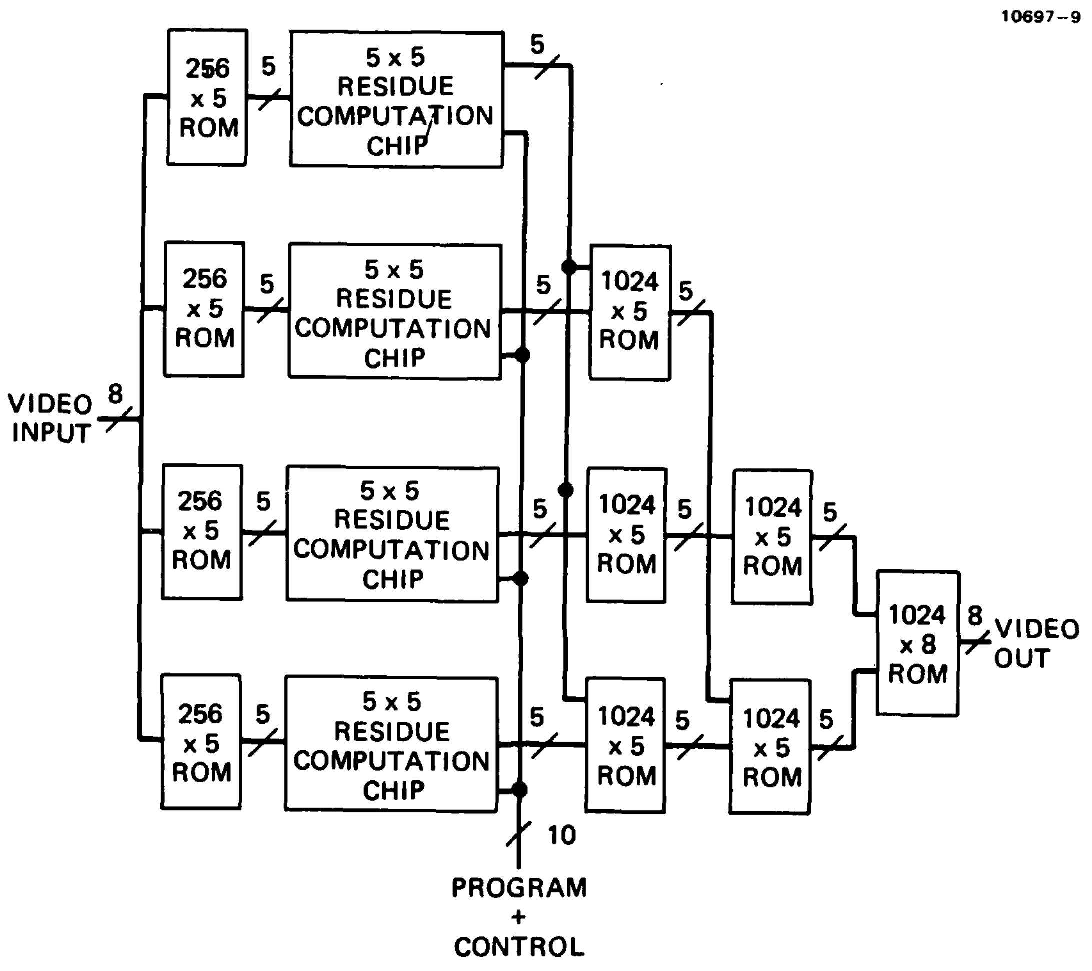


Figure 12. Residue Processor based on 5x5 custom chip.

where y is the output, x_i is the 25 elements in the 5 x 5 kernel, and f_i represents the polynomial functions of a single variable. Each f_i is completely arbitrary and need not have any relations to the other f_i. By selecting subsets of the f_i to be identical to zero, we can program the processor to perform point transforms, 1-D transforms of any size up to 5 x 1, and 2-D transforms of any size up to 5 x 5. Table 3 lists some of the functions that can be performed by this processor.

H. UNIBUS INTERFACE

The processor, as mentioned before, is designed to accept data at 100 nsec intervals. The reason for this high speed design is to allow real-time stand alone operation. This means, however, that when the processor is used as a peripheral device attached to a general purpose computer, the data transfer will be limited by the memory cycle of the general purpose computer and not by the processor speed. Therefore, to get optimal use of the processor, we need the fastest type of transfer available between the processor and the main memory, where the data to be processed will reside. The Direct Memory Access (DMA) type of transfer is the fastest type of transfer that a general purpose computer can support, since it does not require processor intervention. For DEC UNIBUS applications, the fastest data rate one could expect is approximately 1 MHz.

The type of interface we design should then be able to provide a DMA transfer capability for both the programming data and the image data. For either type of transfer, it is essential that the interface be controlled to select the memory location from which the data are to be transferred and to select the number of words to transfer. For program data transfers the interface will only be required to transfer one way at any time. The transfer will be to the processor while in a program mode, and from the processor while in a test mode. For image data transfers the interface must be able to transfer data both ways, for input and output. The simplest alternative to handle this bi-directional transfer of data (from a hardware point of view) is to transfer the output data to the same memory location from which the input data came, i.e; write the output image over the input image.

Table 3. Functional Capabilities of RADIUS

Point Operations Polynominal Functions Contrast Enhancement
1 - Dimensional Operations Integer Coefficient Transforms Polynominal Functions
2 - Dimensional Operations Edge Enhancement Statistical Differencing Low Pass/High Pass Filtering Shape Moment Calculations Statistical Moment Calculations Integer Coefficient Transforms Texture Analysis

DEC devices exist that can provide the DMA transfer capability, as well as provide several control lines to the peripheral device to allow multiple transfer modes. One such device is the DEC DR11B UNIBUS parallel interface. Our plan is to use this device to provide the DMA capability and to design a custom interface to permit the specific transfer modes. The arrangement suggested is shown in Figure 13.

The custom interface will need to interpret the control lines from the DR11B and decide if the transfer is for program data or image data. If it is program data the interface will simply pass the data to the 16 program data lines. If it is an image data transfer, the custom interface is more complex. Since it is a 16-bit transfer, the data will contain two pixels. Following the transfer, the interface must first pass one byte to the input data lines and then the next byte. Simultaneously, the interface must load the first output image data into one byte of the 16-bit output data register, and then load the next output data into the other byte of that register. Finally, the output data

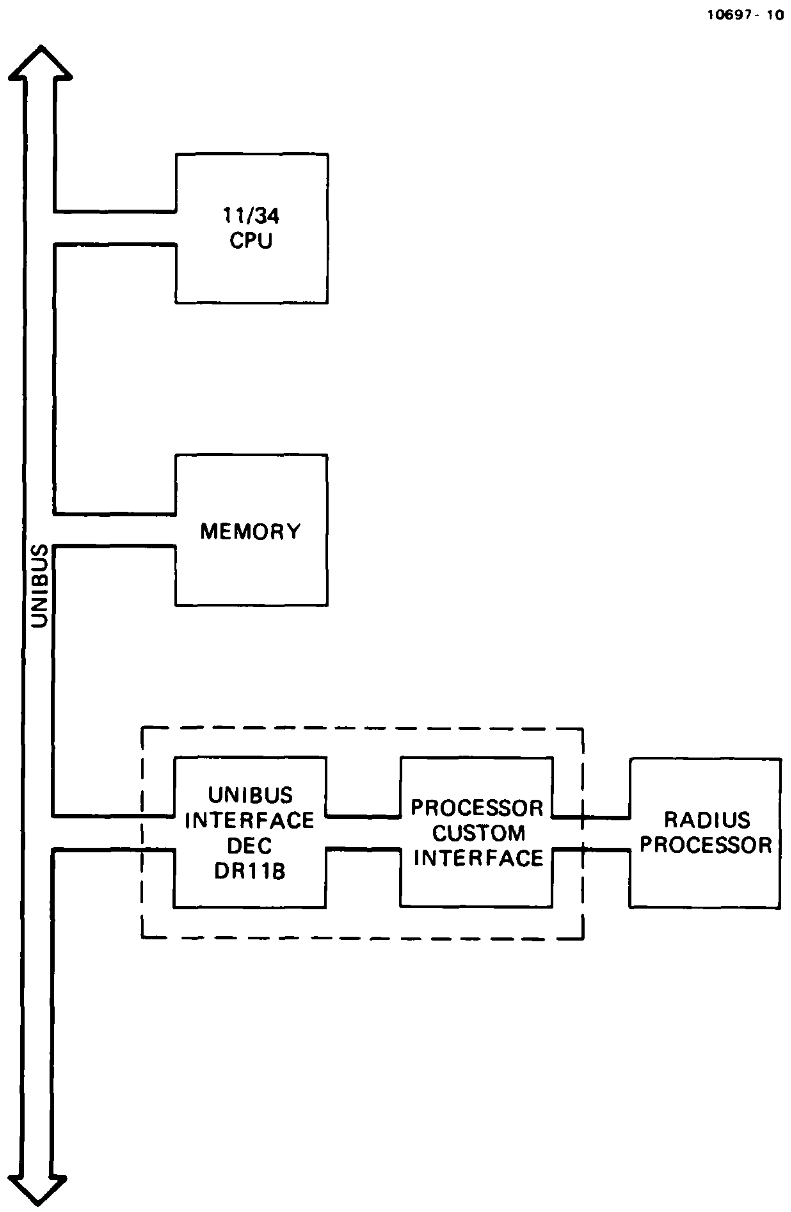


Figure 13. Commercial/custom UNIBUS-processor interface.

registers' contents are transferred to the DR11B which writes it to main memory. A preliminary schematic of a system that can perform these types of transfers is shown in Figure 14.

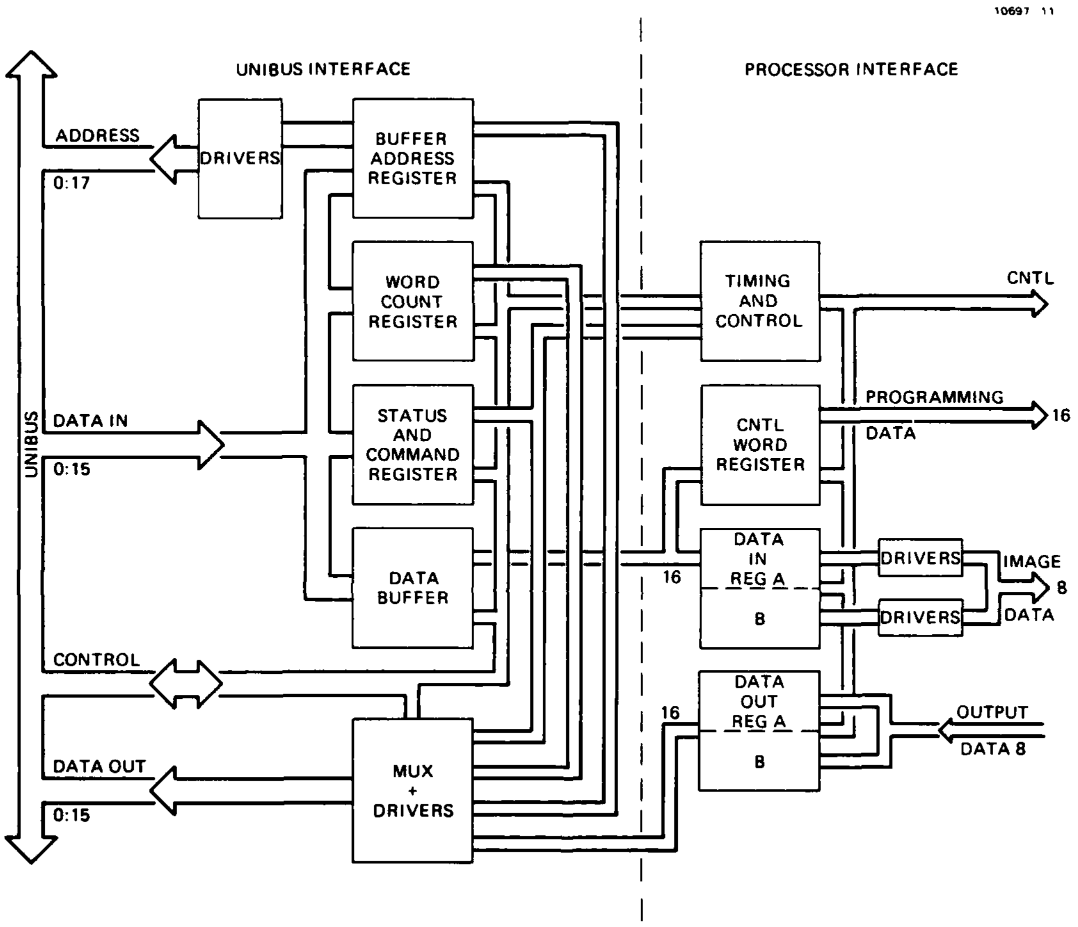


Figure 14. Bus structure of UNIBUS-processor interface.

SECTION 4

SUMMARY

We have described the work undertaken to design VLSI processors for these widely used systems: line-finding, texture classification, and segmentation. From this we believe we can, if required, build the necessary hardware. However, of greater impact, we have identified and started to build a fully software programmable low-level processor for 5 x 5 operations. The circuitry described relies on a special purpose VLSI chip with 6,500 components. Using this, and the interface designed to hook the processor to commercial general purpose machines, most low-level arithmetic operations over a 5 x 5 kernel can be performed.

REFERENCES

1. G.R. Nudd, "Image Understanding Architectures." National Computer Conference, May 1980, Anaheim, CA. AFIPS Conf. Proc 49, 377-390.

2. S.D. Fouse, G.R. Nudd, and P.A. Nygaard, "Implementation of Image Pre-processing Functions Using CCD LSI Circuits". Proc. Society Photo-Optical and Instrumentation Engr. 225, 118-130, Spie Conf. April 1980 Washington, D.C.

3. R. Nevatia and K.R. Babu, "An Edge Detection, Linking, and Line Finding Program," USCIPI Report No. 840, Sept. 1978.

4. K.I. Laws, "Textured Image Segmentation," Ph.D. Thesis, USC, Electrical Engineering Dept., January, 1980.

5. N. Szabo and R. Tanaka, Residue Arithmetic and its Applications to Computer Technology (McGraw-Hill, New York, NY, 1967).

6. R. Ohlander, K. Price, D. Raj Reddy, "Picture Segmentation Using a Recursive Region Splitting Method," Computer Graphics and Image Processing, 1978.

7. S.D. Fouse, G.R. Nudd, V.S. Wong, "Application of LSI and VLSI to Image Understanding Architectures," Proceedings Image Understanding Workshop, April, 1980.

8. S.D. Fouse, V.S. Wong, and G.R. Nudd, "Advanced Image Understanding Using LSI and VLSI," USCIPI Report 990, September 1980, pp. 164-204.

9. A. Huang, "Number Theoretic Processors: A C Array Architecture," Ph.D. Thesis, Stanford, October 1980.

PROGRESS REPORT

April 1981 to October 1981

SECTION 1

INTRODUCTION

It is generally understood that image processing systems have very large computational throughput requirements. (Typically greater than 25 million operations per frame.) If the system is to operate in real-time (30 frames per second [FPS]), then the required throughput is of the order of 1 billion operations per second. Obviously, a special purpose processor is required to achieve real-time performance, and because of the extreme computational requirements it is clear that Image Understanding systems can benefit greatly from the VLSI technologies. To be able to utilize VLSI, however, one must be able to overcome the anticipated high costs of design and test. One way to reduce the cost is to develop a processor which is very modular and can be described in a heirarchical manner, which is how the modern computer design tools will handle the complexity of a VLSI circuit. Another way to reduce the costs is to use regular structures on the chip, such as memory. This will reduce design costs as well as the cost of testing. Finally, the most significant way to handle the cost problem is to make a processor which has application to a wide range of systems. This will allow the cost of the chip to be amortized over a large user base.

For the past year we have been developing an LSI prototype of a VLSI processor which performs arithmetic operations over a sliding 5 x 5 window of an image. The RADIUS (Residue Arithmetic based Digital Image Understanding System) processor was described in the previous USC semiannual report[1]. This processor has several features that make it very well suited to a VLSI implementation, including modularity, extensive use of memory, and application to a large number of image understanding systems currently being developed.

As indicated by the acronym, the processor utilizes the technique of residue arithmetic[2] to perform the computations. The processor converts the incoming binary image data into a residue representation by calculating the MOD or remainder function over multiple, relatively prime bases. The data is then processed in parallel independent channels, one for each base, with identical operations being performed for each base. In each channel the data is processed using modular arithmetic in the respective base. Finally, the data from each base channel are combined to form a binary result.

Our prototype processor uses commercially available read-only-memories (ROMs) for the conversions from binary to residue and residue to binary. For the computation portion of the processor we have developed an LSI nMOS circuit which can perform 5 x 1 local area computations for a single 5-bit base (<32). The processor, which uses 20 of these custom circuits (4 bases, 5 lines per base) is currently programmed to process data in the bases 31, 29, 23, and 19 and can accept 8-bit binary data at a 10 MHz rate.

This report describes the critical aspects of the development of RADIUS and the progress that has been made. In addition, we will describe the status of the essential related projects we have been working on. These include:

- RADIUS-UNIBUS interface
- Applications of RADIUS
- Design Automation
- A Local Area Logic Processor.

SECTION 2

PROGRESS ON RADIUS DEVELOPMENT

The RADIUS development work can be divided into three areas:

- Development of residue custom LSI circuit
- Fabrication of processor board
- System integration.

Each of these are critical in that the system will not operate without the successful completion of the work in all three areas. During the last six months we have made considerable contributions in all three areas.

The development of the residue custom circuit has significantly progressed in the past six months. In June, 1981, the first parts were packaged and tested. Software was developed for testing all of the major components of the circuit, including the input shift registers, RAM's, base latches, and modular adders. A problem was detected with the adders and soon diagnosed to be a design error. At this point we started a re-design of the circuit to correct the error, as well as continuing to thoroughly test out the circuit for functional correctness and operating speed. The chip was run at 2.5 MHz with very clean waveforms. The limit on the speed was due to the bandwidth of the clock generator and clock drivers. The new masks were produced, and the processing of the new lot is due to be completed by the middle of October.

The actual processor consists of a 12 x 14 in. wire-wrap board. The majority of the wire-wrap was complete prior to the arrival of the first lot of chips. The board was tested without the chips, and it was verified that all of the data paths were correct, and that the encoder, adder, and decoder RCMs were correct. When the chips did arrive, they were tested in the processor itself. This provided the benefit of developing the processor programming software simultaneously with the test software. The current status of the processer board is that it is 95 percent complete, with only the fine tuning of the clocks remaining to be done. This will be accomplished when the second lot of chips arrive.

The last area of work instrumental in the development of RADIUS is the integration of the custom chip, processor board, and external control. This work has proceeded in conjunction with the other two areas of effort, since all of the testing was accomplished through the use of the external control. The external control consists of a microcomputer with a 24-bit parallel I/O card. The programs that were written for test and programming have all been written in assembly code and are executed on the microcomputer. This work is essentially complete except for confirming that the system is compatible with the new chips.

SECTION 3

RADIUS-UNIBUS INTERFACE

The RADIUS processor is designed to accept data at 100 nsec intervals, which is fast enough for real-time stand alone operation. This means, however, that when the processor is used as a peripheral device attached to a general purpose computer, the data transfer will be limited by the memory cycle of the general purpose computer and not by the processor speed. The Hughes Image Understanding Installation is based on a PDP 11/34, with a so-called Direct Memory Access module, type DR11B, providing the fastest access. The PDP 11/34 is able to communicate with the computer memory along the UNIBUS lines without intervention by the CPU. About 500,000 words per second can be transferred in this way, which translates to a data rate of 1 Mbyte/sec at the RADIUS processor.

The PDP 11/34 uses a page-addressed memory structure, each page being 32K words in length. In order to store a complete image it is necessary to cross page boundaries by a technique known as dynamic region allocation. Software has been written to do this, which stores a 256 x 256 x 8 bit image buffer ready for processing. Our display unit, a COMTAL, can store two 512 x 512 x 8 bit images, each of which is split into four quadrants giving 8 additional image buffers. It is anticipated that 9 buffers in all should be adequate for the development of most of our Image Understanding algorithms.

The dynamic region allocation and direct memory access techniques will provide a string of 8-bit values, originally generated by a raster scan of the image. We will include in the processor interface the means for generating the two dimensional kernel. This kernel generation function is most easily envisioned as a series of shift registers. For a 5 line kernel, four shift registers are required, each one containing as many elements as there are pixels on a line. However, since the system is being designed with variable line lengths, a variable length shift register would be difficult to implement. For this and other reasons (including component availability, price, performance, reliability, etc.) Hughes has developed a line delay system using random access memory. A system of address counters and crossbar switches is used to both load and retrieve the data on a line by line basis, and also to access the 5 x 5 kernel itself.

Another design issue in the computer-to-RADIUS processor interface is that of gaining access to the look-up tables carried in each of the 20 custom residue chips. This could be done either with a separate interface module to the PDP 11/34 or by using some of the existing control lines on the DR11B DMA module. We will take the latter course of action to avoid unnecessary clutter on the PDP 11/34 backplane. This will, however, necessitate some extra switching on the interface module so that data from the DR11B can be routed to either the four line delay or to the look-up tables. For reasons described below, we indend to pass information to and from the RADIUS processor through look-up tables composed of 256 x 8 bit RAMS. This will make it possible to apply non-linear operations on a pixel by pixel basis, and will also control the dynamic range of signals fed to the RADIUS processor.

SECTION 4

APPLICATIONS OF RADIUS

The primary motivation for developing this processor was to do 5 x 5 convolutions for such applications as edge enhancement, statistical differencing, low/high pass filtering, statistical moment calculations, integer coefficient transforms, and texture analysis. However, the processor is capable of performing a much wider range of computations. The reason for this flexibility is due to the fact that we used a look-up table to perform a unary operation, and that the table is completely programmable. The general form of the computation that can be performed by the processor is

$$y = \sum_{i} f_i(x_i),$$

where y is the output, x_i is the 25 elements in the 5 x 5 kernel, and f_i represents any allowable residue arithmetic computations.

The exact nature of what constitutes an "allowable residue arithmetic computation" is an interesting question, and we are performing ongoing studies to seek a solution. Basically, the integer operations of addition, subtraction, and multiplication are allowed, but division is not, since the result of a division is not, in general, an integer. In the residue arithmetic as implemented on the RADIUS machine, any number is uniquely represented by an array of four residues in bases 19, 23, 29, and 31. If any addition, subtraction, or multiplication operation is performed on all four residues (modulo the base in question), the result is equivalent to performing the same operation on the input number. The largest number that can be represented, M, is given by the product of the bases, in this case 392,863. Overflow cannot occur in any one base since the answer is always taken modulo the base, but it is possible for the output number to exceed M. In this case, the computed result appears modulo M, giving a 'wrap around' effect similar to that in binary arithmetic.

However, there is no convenient way to determine that this overflow has occurred, necessitating considerable care in developing an algorithm to generate $f_i(x_i)$. (It is imperative that the final result of the algorithm does not exceed M.) Nevertheless, an advantage of residue arithmetic is that intermediate values in a calculation can be arbitrarily high, and the algorithm can be arbitrarily complex. This arbitrary complexity is a very powerful feature of residue arithmetic, and it arises because the results are stored in a look-up table. However, because much computer time is needed to generate the tables themselves, the computations are still performed on image data at the full 10 MHz rate.

Since the RADIUS processor can evaluate any arbitrary integer coefficient polynomial and many useful operations can be approximated by polynomial functions, we have decided to investigate this approach. The polynomial may be of arbitrarily high order with arbitrarily large coefficients, although the coefficients must be integer. In particular, no coefficient can be less than one, so that in a high order term a large input number raised to a high power may exceed the maximum, M. This is not a problem if other terms in the polynomial are sufficiently negative to reduce the overall result to less than M (we are working on polynomial curve-fitting techniques to achieve this). Figure 1 shows an approximation to the function, $y = a\sqrt{x}$, useful as part of the Sobel operator. As may be seen, an accurate fit is obtained with a cubic polynomial everywhere except at the origin, where it is possible to improve the fit, if necessary, by adjusting the polynomial coefficients. There is a tradeoff between accuracy of the fit and the maximum allowable input value, but this is rapidly improving as we improve our curve fitting procedure. The preliminary results shown in Figure 1 were produced simply by performing a least squares fit and then rounding the coefficients to the nearest integer value, but this is far from being the optimal technique. It is possible to fit a polynomial curve to practically any desired function, so that such operations as contrast enhancement, thresholding, etc., become possible. Furthermore, it is possible to perform a different function for each element of the kernel, so that the RADIUS processor is expected to open a new area in convolution-type processing algorithms.

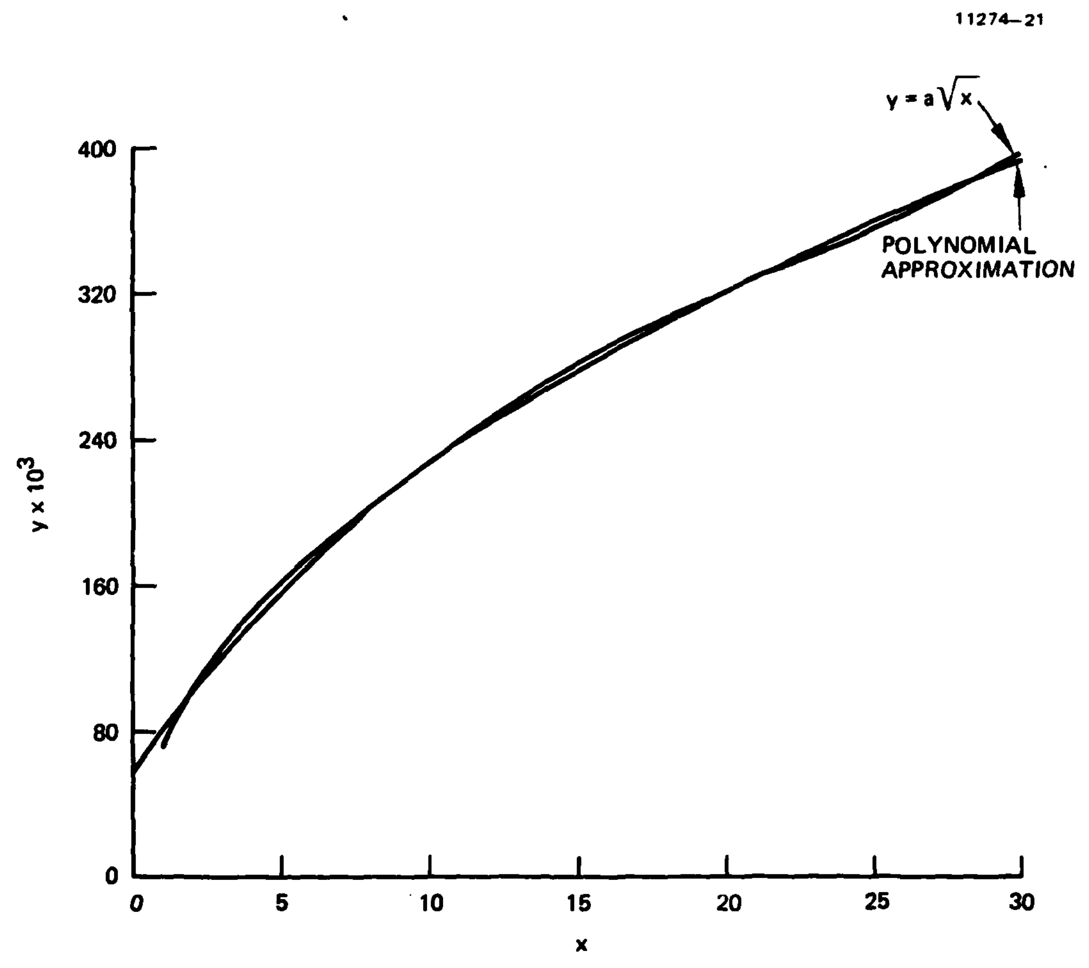


Figure 1. Scaling factor chosen to utilize dynamic range of a residue processor.

SECTION 5

DESIGN AUTOMATION

This work is not directly part of the I.U. effort but will be of great benefit in terms of cost and speed of our designs. We have set ourselves the goal of investigating and implementing a design automation system capable of designing large VLSI chips in an efficient and reliable manner. If the predicted downscaling of devices and increase of chip density materializes in the near future, we will need much better design tools to organize and design VLSI chip systems, since we see that design productivity and system complexity will be the bottlenecks in implementing VLSI systems.

To help us develop these tools for VLSI design, we have at our disposal several computer systems and devices. These include a VAX 11/780, where we are developing most of our software, an AVERA IC designer, which is a complete IC design system in itself, a PDP 11/34, which is used as a controller, and several plotters and graphics terminals. We also have access to an Amdahl 470 processor and a PDP 10 computer for running simulations and for program development.

The AVERA unit is a self-contained IC design system, including layout graphics and symbolic representation of designs. It includes dual floppy disks where designs and system programs are kept, a 17 in. black and white CRT, a keyboard, and digitizing tablet. The capabilities include symbolic recognition of commands and up to 64 levels of design representations. The design output can be in either CALMA GDS II or CIF format.

We are also investigating the design automation approaches being pursued at the Universities. One example is a STICKS-type design package called CABBAGE, developed at UC Berkeley. It enables a designer to use symbolic representations to formulate his designs, and as a final step, compacts the design to minimum geometry while observing the programmed design rules. Another approach being taken at Caltech and MIT is to specify the designs in a structured and algorithmic way. Standard VLSI components such as PLAs and ALUs can be designed in this way with alterable parameters controlling the size and configuration of the component. Other groups were looked at during the testing and simulation of complex VLSI system designs. In order to design error-free systems, it is necessary to be able to simulate complex chip designs. Research on this is being done at MIT, where a program for logic simulation called MOSSIM has also been developed.

SECTION 6

A LOCAL AREA LOGIC PROCESSOR

RADIUS has wide application to image understanding and can perform the vast majority of arithmetic operations required. There is, however, a need for additional high-speed processing at the pixel level for operations such as those requiring logical decisions. Examples of the need for this type of processing are operations such as edge thinning, edge tracing, and region formation. Recognizing the need for this type of high speed processing, we are working to define and develop a logic processor to complement RADIUS.

The first step in the development of the logic processor is to define an instruction set. This set of instructions should allow a wide range of functions to be performed on an image in single and multiple passes. The processor will access a local neighborhood of each pixel and will produce an output based on a comparison of the neighborhood to a template or a set of conditions. This concept is illustrated for three image frames in Figure 2. The development of instructions or constraints required for the logic processor are not difficult to develop and have been generated for processors such as PICAP[3]. What we shall aim for is an efficient set of operations to allow high speed performance, matched to the RADIUS machine. The type of neighborhood that is accessed, the types of conditions that can be specified, and the types of mappings from the combination of local neighborhood and conditions to an output pixel determine the instruction set parameters and specify a minimum capability for the processor.

Once the processor instruction set is defined, we will determine an appropriate architecture. As with the RADIUS processors we will look towards architectures that will be suited to VLSI implementation. For the same reasons that motivated the design of RADIUS, we will probably utilize look-up tables to perform some of the operations, making good use of memory structures which are easily designed.

A high level block diagram of a processor which could probably perform the range of operations required of a logic processor is shown in Figure 3. This processor is comprised of logic to form the local neighborhood, which can be controlled to perform comparisons on the data in the neighborhood, and two

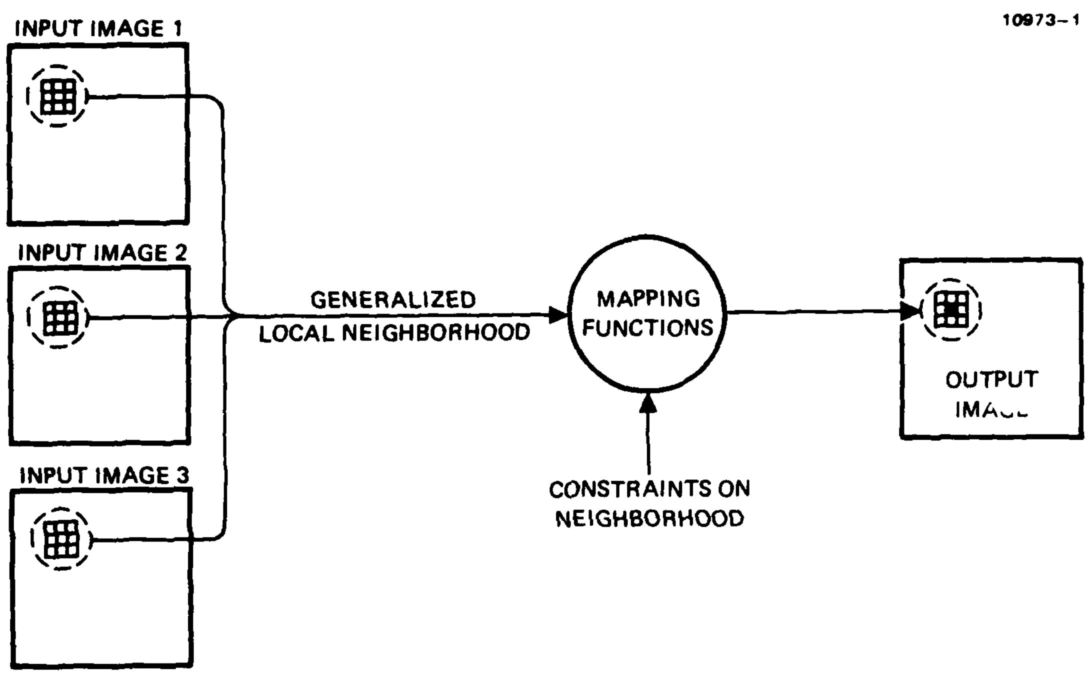


Figure 2. Inputs and outputs for logic processor.

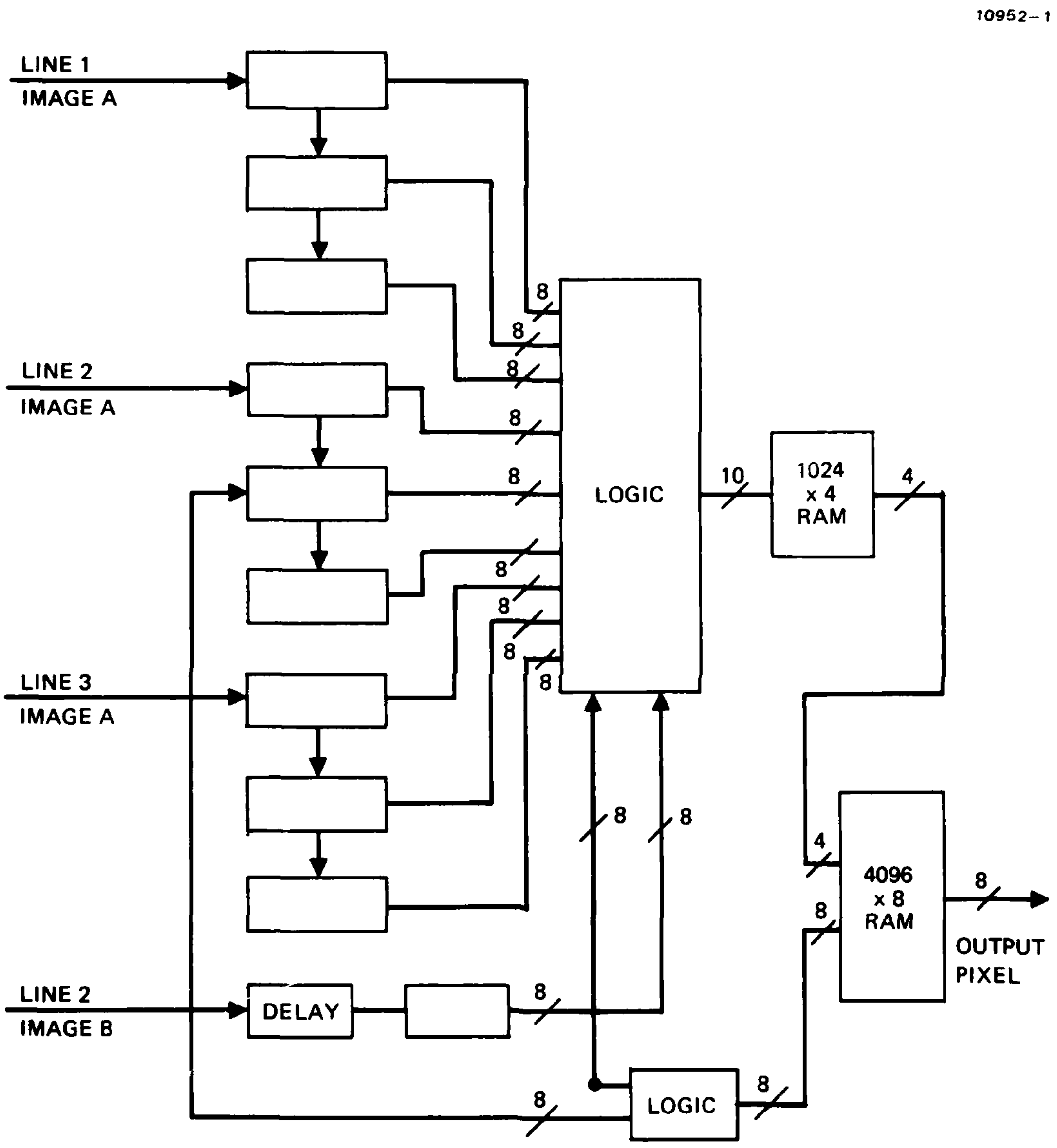


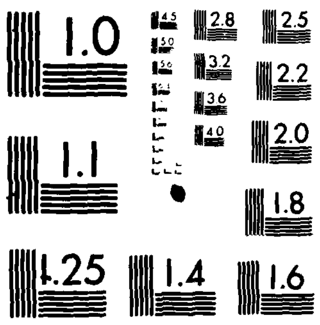
Figure 3. An architecture for a local area logic processor.

look-up tables to provide the mapping from the results of the comparisons and the center pixel to an output pixel. The look-up tables, in addition to making a chip easy to design, will also give the processor a great deal of flexibility. As a benchmark example, we have investigated the median function. Using this architecture, the processor would be capable of performing a median calculation of some local neighborhoods with only 8 passes. This is just an indication of the power a table-driven processor could provide.

AD-A116 237   HUGHES RESEARCH LABS MALIBU CA   F/G 9/2
ADVANCED IMAGE UNDERSTANDING.(U)
DEC 81   G R NUDD, S D FOUSE   F33615-80-C-1080
UNCLASSIFIED   AFWAL-TR-82-1095   NL

2 2

END
DATE
FILMED
7-82
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

will also be developed

SECTION 7

SUMMARY (are discussed)

in report

We have described ongoing work on the development of RADIUS, a real-time image understanding system which is well suited to VLSI. In addition to the development work we have described, we are working on several related tasks including developing an interface between a PDP 11/34 and the RADIUS processor, investigating further applications of residue arithmetic to image understanding, and developing an integrated design automation capability that will allow us to design and simulate LSI and VLSI circuits.

~~Our~~ future work will include further development of the RADIUS processor and the development of the local area logic processor we described. When they are both complete we will have an integrated pixel level processor that can perform a wide range of functions in real-time and many more functions in near real-time. We will also develop an interface to the PDP 11 for the integrated RADIUS-LOGIC processor, and this will allow all pixel operations to be performed at high speed, reducing greatly the CPU time needed for image understanding programs.

REFERENCES

1. S.D. Fouse, G.R. Nudd, G.M. Thorne-Booth, P.A. Nygaard, and F.D. Gichard, "A Residue Based Image Processor For VLSI Implementation," USCIPI Report 1010, March, 1981, pp. 73-98.

2. N. Szabo and R. Tanaka, _Residue Arithmetic and its Applications to Computer Technology_ (McGraw-Hill, New York, 1967).

3. B. Kruse, "System Architecture for Image Analysis," Chapter 7 of _Structured Computer Vision_, edited by S. Tanimoto and A. Klinger (Academic Press, 1980).

8

DTIC